Departamento de Informática

PROYECTO FIN DE CARRERA
Ingeniería Técnica en Telecomunicaciones.
Especialidad en Sistemas de Telecomunicación.

Módulo de análisis del tráfico y energía de aplicaciones de redes de sensores simuladas con Avrora

Autor: María del Carmen Arias Escolar
Tutor: Alejandro Calderón Mateos
Codirectora: María Soledad Escolar Díaz

Leganés, septiembre de 2015

Título: Módulo de análisis del tráfico y energía de aplicaciones de redes de sensores simuladas con Avrora.

Autor: María del Carmen Arias Escolar
Director: Alejandro Calderón Mateos
Codirectora: María Soledad Escolar Díaz

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

"Hay una fuerza motriz
más poderosa que el vapor,
la electricidad,
y la energía atómica:
la voluntad"

Albert Einstein

Agradecimientos

A Soledad Escolar, que durante tanto tiempo me ha sabido guiar, apoyar y animar para la realización de este proyecto. Sin ti, nada de esto hubiera sido posible.

A Alejandro Calderón Mateos, por acoger este proyecto.

A Gloria, mi hermana, por todo el tiempo que ha invertido ayudándome para poder conseguir mis metas.

A mis padres, Manuel y Mari Carmen, por estar siempre conmigo, por no dudar nunca de mí. Os quiero.

A Luis, mi marido, por caminar a mi lado y nunca separarse de mí, por sacarme una sonrisa cuando más difícil creo que es sonreir, por su infinita paciencia. Decir que te quiero se queda corto para expresar lo que siento por ti.

Y a Paula, mi hija. Todo es por ti y para ti. Ahora la vida tiene sentido. Gracias por existir.

Resumen

Este Proyecto Fin de Carrera se centra en la interpretación y posterior representación gráfica de la información obtenida tras realizar una simulación de una red de sensores inalámbricos (del inglés Wireless Sensor Networks, WSN), utilizando el simulador Avrora.

Una Red de Sensores Inalámbrica, es una red basada en pequeños dispositivos denominados nodos sensores o motes con capacidad de procesamiento y comunicación, y que integran un conjunto de sensores capaces de muestrear distintas variables en su entorno, como por ejemplo temperatura, ruido, o humedad. Los nodos sensores monitorizan un determinado fenómeno de interés, y cooperan entre sí para reenviar los datos recolectados a un computador, denominado estación base, con mayores capacidades para su procesamiento offline.

La simulación de las aplicaciones WSN es un proceso crítico que debe llevarse a cabo antes de realizar el despliegue en un escenario real. Avrora es uno de los simuladores de redes de sensores inalámbricos más utilizado, que dispone de varios monitores que permiten controlar y gestionar aspectos de la simulación tales como la energía consumida por los nodos que forman parte de la red, el tráfico que cursan, el uso de las distintas memorias del microcontrolador, etc. Los datos asociados a estos monitores se presentan al usuario en modo texto en la consola de simulación, por lo que la interpretación de la información obtenida es tediosa y compleja.

Este proyecto tiene como objetivo dotar de una interfaz gráfica, programada en Java, al simulador Avrora, que facilite la lectura y la comprensión de los datos producidos por el simulador. La interfaz gráfica se ha desarrollado en particular para los monitores de paquetes (*packet),* que muestra el tráfico cursado por la red durante la simulación y el monitor de energía (*energy)* que muestra la energía consumida por los nodos sensores tras la simulación. La interfaz gráfica que se ha desarrollado muestra de una manera sencilla los datos más relevantes de tráfico y energía que proporciona Avrora tras una simulación de redes de sensores. El módulo desarrollado analiza los resultados de Avrora para proporcionar información adicional interesante que permita evaluar el rendimiento del sistema, como por ejemplo la cantidad de paquetes que envía y recibe cada nodo, diferenciar los tipos de paquetes que envía cada nodo, los caminos de enrutamiento de los paquetes, (nodo destino y origen, respectivamente), el porcentaje de tráfico que representan con respecto al total de tráfico de la red, entre otras informaciones. Esta información nos permite estimar mejor el comportamiento y el rendimiento de la red de sensores que se pretende desplegar.

Palabras clave: Wireless Sensor Networks, Avrora, Java, simulación.

Abstract

This Ending Career Project is focused on the analysis and visualization of the information provided after a Wireless sensor Network (WSN) simulation with the Avrora simulator.

A WSN is a network composed of tiny devices, also called sensor nodes, which have capabilities of processing and communication, and a set of sensors able to sample different environment variables, such as temperature, noise, or pressure. The sensor nodes are intended to monitor a phenomenon of interest, for which sample their sensors and transmit the data collected towards an external computer, typically called base station, with major capabilities for the futher analysis of the data.

Simulation of WSN applications is a critical proccess before deployment on real scenarios. Avrora is one of the most used simulators of WSN, with several monitors which allow controlling and managing simulation aspects such as the energy consumption of each node, network traffic and usage of the memory. The resulting information from the execution of these monitors is showed in text mode in the simulation console, so it is complicated to interpret the data.

This Project has as primary objective to provide a graphical interface, developed with the programming language Java, to the Avrora simulator. The interface will ease the users the reading and understanding of the data generated by the simulator. This interface has been developed specifically for monitors *packet* and *energy*, which deal with the network traffic and the energy consumption, respectively. The graphical interface shows in a simple and easy way the most relevant data of the communication among nodes and the energy consumption of each node, after an Avrora simulation. Besides this, the module that we have developed analyzes the results provided by Avrora in order to compute additional information that enables to evaluate the behavior of the network. For example, we can compute the amount of data packets sent and received for each node, the type of the packets, the network topology, the paths between sources and destinations. This information enable to us to estimate better the behaviour and performance of the WSN that is supposed to be deployed.

Keywords: Wireless Sensor Networks, Avrora, Java, simulation.

Índice general

Índice de Ilustraciones

Índice de Tablas

1. Introducción

Una red de sensores inalámbrica, del inglés, ***W**ireless **S**ensor **N**etworks* (WSN) es un sistema distribuido en el que algunos de sus nodos (o todos), denominados nodos sensores o motes son capaces de interactuar con el entorno físico. Estos nodos están equipados con distintos tipos de sensores y/o actuadores que les permiten monitorizar una gran cantidad de fenómenos, mediante la medición de parámetros y variables físicas, tales como temperatura, la humedad, la luz, la presión o el ruido. Los nodos disponen de capacidad de procesamiento, almacenamiento y comunicación que les permiten ejecutar pequeñas aplicaciones que básicamente consistirán en muestrear los sensores y enviar los datos así obtenidos hacia una estación base, o computador con mayores capacidades, donde puedan ser almacenados, procesados y estudiados a posteriori.

Las redes de sensores inalámbricas están actualmente en continua expansión. En el campo del hardware, los dispositivos sensoriales tienden a ser cada vez más pequeños por el efecto de la miniaturización pero con mayores capacidades. Disponen cada vez de más autonomía, gracias al uso de baterías más eficientes o el empleo de fuentes de energía renovables como por ejemplo células solares. En el campo del software, el potencial de las aplicaciones de redes de sensores permite que sean usadas en cada vez mayor número de dominios y escenarios. Como muchas tecnologías, las WSN comenzaron su desarrollo en el ámbito militar. Ahora, tienen multitud de aplicaciones civiles tales como el control medioambiental, vigilancia en procesos industriales, la automatización del hogar, control de procesos agrícolas, etc.

Uno de los desafíos más importantes en las WSN es el consumo energético. La mayoría de las plataformas comerciales vienen equipadas con un par de baterías para permitir la operación del nodo. Esto quiere decir que el tiempo de vida de un nodo sensor viene limitado por el tiempo que tarda en agotarse sus baterías, lo cuál depende básicamente del uso intensivo que haga de sus recursos de hardware. En una red de sensores, en el peor caso, el tiempo de vida de la red está vinculado al primer nodo que agota sus baterías. Por este motivo, es muy importante controlar el consumo energético de los nodos para garantizar el correcto funcionamiento de la red. Es por esto que en ocasiones es muy adecuado simular el comportamiento de las redes, con el fin de estimar el comportamiento y funcionamiento de la red en el entorno desplegado y así conocer de antemano cuáles son los puntos problemáticos de la red, ya sea por ser cuellos de botella de tráfico, provocando congestión en la red, o por exceso de consumo de baterías, lo cual haría que los nodos dejaran de funcionar y se tuviera que reestructurar la topología de la red y cambiar las tablas de enrutamiento de los nodos sensores para que los paquetes enviados llegaran, finalmente, a la estación base.

Este Proyecto Fin de Carrera se enmarca dentro de la simulación de redes de sensores inalámbricas para la estimación de la energía consumida por parte de los nodos de la red y el análisis de tráfico. A continuación plantearemos el problema, describiremos nuestra propuesta de proyecto que trata de resolver esta problemática y los objetivos del proyecto.

1.1 Descripción del problema y motivación.

Uno de las herramientas más utilizadas en el ámbito de las redes de sensores es el simulador Avrora. Avrora es un potente simulador para aplicaciones de redes sensores

inalámbricas desarrolladas sobre el sistema operativo TinyOS 1.x/2.x. El simulador Avrora se ha desarrollado en Java, lo que permite que pueda ser ejecutado en cualquier entorno que tenga instalada la máquina virtual de Java JVM. Entre sus funciones más destacables se ha de destacar que proporciona un conjunto extenso de monitores para analizar en detalle la red a simular, topologías, la energía consumida por cada componente de hardware y estado, y el tráfico de los paquetes enviados y recibidos, entre otros.

La información obtenida durante la simulación de la red de sensores inalámbricos se presenta al usuario final en modo texto en la consola de salida desde donde se haya lanzado la ejecución de Avrora. La salida por pantalla en modo texto, hace muy difícil sino imposible la lectura y el análisis de los datos en tiempo real, por lo que es necesario invertir mucho tiempo para llegar a saber qué tipo de tráfico se ha movido por la red, cuáles son las comunicaciones entre los nodos que la forman y por consiguiente, la topología de la red de sensores, así como los valores de consumo energético de los nodos.

Este Proyecto Fin de Carrera pretende resolver este problema por medio del desarrollo de un módulo que pueda ser fácilmente integrado con el simulador Avora y que presente los datos de la simulación de las aplicaciones de redes de sensores de una manera gráfica. Gracias a este módulo, se podrá conocer de manera sencilla y de un simple vistazo el número de paquetes enviados/recibidos por cada nodo de la red, la energía que consume cada nodo, por cada componente y estado y las interacciones entre los nodos de la red. Además, será posible conocer la topología de la red, tanto antes de comenzar la simulación como tras la finalización de la misma, dado que la propia ejecución dinámica del protocolo de enrutamiento podría modificarla.

1.2 Objetivos del proyecto

Tal y como se ha descrito en el apartado anterior, el objetivo primario de este proyecto es crear un módulo que se integre con la herramienta Avrora y que lo dote de una interfaz gráfica en la que se pueda ver cuál ha sido el comportamiento de la red de sensores que se ha simulado.

Los hitos que se han de cumplir para la consecución de este proyecto y que se desarrollarán a lo largo del documento son los siguientes:

- **Estudio del estado del arte en redes de sensores inalámbricas**. Este estudio de las redes de sensores ocupa un espacio reseñable en la elaboración de este proyecto y sin él no hubiera sido posible la elaboración del mismo, por lo que nos los hemos planteado como un objetivo del mismo. Se ha realizado un concienzudo análisis sobre qué son las redes de sensores, y su evolución desde la red *SoSuS* (**So**und **Su**rveillance **S**ystem) que apareció en 1949, hasta nuestros días, en su uso de aplicaciones que se quieren desplegar en el planeta Marte. Se han desarrollados las características de estas redes, (tales como seguridad, la tolerancia a fallo, su infraestructura, la energía, sus costes, etc). Como punto final al estudio general de las redes de sensores inalámbricos, se ha analizado el hardware que se puede usar en el despliegue de las WSN como nodo sensor (BT Node3, las motes de la familia Mica como Mica2 o MicaZ, TinyNode, etc), placa de sensores (MDA320 o las de la familia MTS como pueden ser MTS300, MTS310, etc), y como nodo gateway (los netBridge o los MIB510 y MIB520 por ejemplo), así como el software que las motes pueden tener embebido (TinyOS, Contiki, Mantis, etc).

- **Control de los parámetros de entrada de la simulación**. Se podrá conocer en todo momento cuáles han sido los parámetros que se han seleccionado para la simulación, así como los ficheros de topología de red utilizados.
- **Simulación de las aplicaciones y análisis del tráfico** cursado por cada uno de los nodos, incluido el nodo gateway.
- **Monitorización del consumo energético** del módulo de radio de los nodos sensores y del nodo gateway, tanto en recepción como en transmisión.
- **Desarrollo de un módulo en Java que permita la representación gráfica** del estado de la red antes de comenzar la simulación, la monitorización de la simulación y la representación después de finalizar la simulación.

1.3 Estructura del documento

Este documento se estructura en siete capítulos más una sección de bibliografía usada para el desarrollo de este proyecto fin de carrera. A continuación, se presentará una breve descripción de cada uno de estos capítulos:

- En el primer capítulo, llamado "*Introducción"* se plantea una visión general del trabajo realizado a lo largo de este proyecto fin de carrera, y las motivaciones para llevarlo a cabo. De igual modo, se incluyen los objetivos que se desean alcanzar. Al finalizar este capítulo, se incluye un glosario con los términos más importantes usados, con el fin de facilitar la lectura y comprensión del documento.
- El segundo capítulo es "*Estado de la cuestión"*, donde se describe qué son las redes de sensores inalámbricas, sus elementos y arquitectura, y los sistemas operativos diseñados específicamente para estos dispositivos, atendiendo a sus restricciones en cuanto a cómputo, comunicación y energía. Se explica también algunas de las muchas aplicaciones donde las WSN podrían tener un claro impacto.
- El tercer capítulo, denominado "*Especificación de requisitos"*, se centra en las necesidades funcionales, de apariencia, de rendimiento y de sistema necesarias para el buen funcionamiento de la interfaz gráfica que se pretende desarrollar en este Proyecto Fin de Carrera.
- El cuarto capítulo, "*El simulador Avrora"* explica el funcionamiento de Avrora, monitores, y los parámetros necesarios para poder simular una red de sensores. También se centra en la descripción de los resultados obtenidos durante la simulación que usa dos de los monitores más importantes de Avrora, el monitor de *paquetes* y el monitor de *energía.*
- El quinto capítulo, "*Diseño"*, presenta el diseño de la interfaz gráfica desarrollada para mostrar los datos analizados durante la simulación, así cómo la estructura que tienen las clases que se han implementado para conseguir el diseño final de la interfaz gráfica.
- En el sexto capítulo, "*Implementación*", se describen cada una de las clases que componen el paquete *AvInterfaz,* desarrollado durante la elaboración de este proyecto. En cada una de las clases, se da una visión detallada de sus funciones y métodos principales así como sus atributos y variables utilizadas más relevantes.
- En el séptimo y último capítulo, "*Conclusiones",* se muestra una visión global sobre el trabajo realizado y se evalúa el cumplimiento de los objetivos planteados al inicio de este proyecto.

1.4 Glosario

Avrora

Avrora es un simulador para aplicaciones de redes de sensores basadas en TinyOS, desarrollado en la Universidad de California (Los Ángeles) y formado por un conjunto de herramientas diseñadas para la simulación y análisis de programas implementados para los microcontroladores de la familia Mica e Iris. Dispone de un conjunto de monitores que permiten analizar los recursos de la aplicación simulada (energía, ciclos de CPU consumidos durante cada proceso, etc). Avrora es un proyecto de código abierto por lo que además puede extenderse fácilmente e incluir nuevas funcionalidades.

Enrutamiento

También conocido como encaminamiento, se trata de la función de buscar un camino entre todos los posibles caminos elegibles en una red, en base a alguna métrica como por ejemplo el camino mínimo, más eficiencia energética, menor retraso, etc. Dado que se trata de encontrar la mejor ruta posible, lo primero será definir qué se entiende por mejor ruta y en consecuencia cuál es la métrica que se debe utilizar para medirla.

Gateway

Dispositivo o pasarela que permite la interconexión entre dos interfaces de red. En este proyecto, el gateway permite la comunicación entre la red de sensores y una estación base o computador que suele disponer de una conexión directa a una red TCP/IP.

Java

Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

MicaZ

Es uno de los modelos de mote más conocidos y usados. Es distribuido por la empresa Crossbow Technology, empresa tecnológica que surge de la Universidad de Berkeley (California).

Mote

Se denomina mote (o nodo sensor) al dispositivo con sensores y capacidad para realizar mediciones y transmitirlas, que colabora con otros para formar una WSN.

nesC

Es un dialecto del lenguaje de programación C optimizado para las limitaciones de memoria de los nodos sensores. Existen varias herramientas que completan y facilitan su uso, escritas en su mayoría en Java y en Bash. Otras herramientas y librerías asociadas están escritas principalmente en C. Es un lenguaje orientado a componentes y está especialmente diseñado para programar aplicaciones sobre redes de sensores, en particular en el sistema operativo TinyOS.

Sensor

Transductor o dispositivo cuyo fin es recibir estímulos y transformarlos en información, permitiendo así medir fenómenos físicos.

Simulador

Dispositivo o aparato que emula un fenómeno, el funcionamiento real de otro aparato o dispositivo o las condiciones de entorno a las que están sometidos una máquina, aparato o material.

Topología

Es la forma en que está desplegada la red, sea en el plano físico o lógico.

TinyOS

Es un sistema operativo de código abierto basado en componentes para redes de sensores inalámbricas. Está escrito en el lenguaje de programación nesC como un conjunto de tareas y procesos que colaboran entre sí. Está diseñado para incorporar nuevas innovaciones rápidamente y para funcionar bajo las importantes restricciones de memoria que se dan en las redes de sensores. TinyOS está desarrollado por un consorcio liderado por la Universidad de California en Berkley en cooperación con Intel Research.

WSN

Wireless Sensor Network o Red de Sensores Inalámbrica, es una red basada en pequeños dispositivos dotados de sensores y capacidad para realizar mediciones, que cooperan entre sí para monitorizar un fenómeno.

ZigBee

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radios digitales de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal (Wireless Personal Area Network, WPAN). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías.

2. Estado de la Cuestión

En este capítulo explicaremos qué son las redes de sensores inalámbricos, cuál ha sido su evolución, cuáles son sus componentes y cuáles son las aplicaciones más comunes de estas redes. También veremos los sistemas operativos que se utilizan para el funcionamiento de los nodos sensores y las tecnologías de software usadas.

2.1 Redes de Sensores Inalámbricas

El desarrollo de las redes de sensores inalámbricas (del inglés *Wireless Sensor Networks*, WSN), y su implantación en aplicaciones comerciales y militares ha sido posible gracias a todos los avances en comunicaciones inalámbricas, el desarrollo en la micro-fabricación e integración de circuitos y la evolución de los microprocesadores empotrados. Este desarrollo ha propiciado la aparición de dispositivos sensores de reducido tamaño, bajo coste, consumo energético mínimo, con capacidad de procesar información del entorno de forma local y con capacidad de comunicación inalámbrica.

Una red inalámbrica de sensores está formada por una gran cantidad de los dispositivos anteriormente descritos, también llamados *motes*, nodos sensores, o simplemente nodos, capaces de recoger información de su entorno gracias a que están equipados con uno o más dispositivos transductores o sensores de distinto tipo (por ejemplo, humedad, luz, temperatura, presión, aceleración, etc.). La comunicación entre los nodos se realiza de forma inalámbrica, dentro de su rango de cobertura. Es por ello por lo que en estas redes aparece la figura de los nodos enrutadores, que son aquellos encargados de encaminar los datos hacia un dispositivo especial, denominado gateway, encargado de enlazar la red de sensores con un tercer dispositivo, típicamente de mayor capacidad, denominado estación base. El gateway se conecta a la estación base vía un protocolo serie (e.g. RS-232, USB) o vía Ethernet, lo cual permite que en muchas ocasiones, la estación base se encuentre lejos del área de cobertura de los nodos sensores.

2.1.1 Historia de las WSN

El hombre siempre ha sentido la necesidad de predecir los cambios relevantes de su entorno, bien de naturaleza física como atmosférica. Esta inquietud provoca que se comiencen a usar sensores electrónicos para medir el entorno y de este modo poder calcular tendencias.

La verdadera evolución de las redes de sensores comienza, al igual que otras tecnologías, en iniciativas militares y de defensa. SOSUS (**SO**und **SU**rveillance **S**ystem, *sistema de vigilancia sónica*) (01) es considerada la primera red de sensores conocida. Esta red estaba compuesta por un conjunto de boyas sumergidas, instaladas en el paso GIUK (en la línea imaginaria que une Reino Unido con Islandia y Groenlandia) durante la guerra Fría, con el fin de detectar submarinos soviéticos usando sensores de sonido. Sin embargo, los dispositivos sensoriales utilizados en este proyecto están aún lejos de parecerse a los nodos sensores tal y como los conocemos en la actualidad.

En 1980, DARPA (**D**efense **A**dvanced **R**esearh **P**rojects **A**gency, *Agencia de Investigación de Proyectos Avanzados en Defensa* de Estados Unidos) comenzó la investigación moderna en redes de sensores con el proyecto DSN (**D**istributed **S**ensor **N**etworks, *redes distribuidas de sensores*) (02).

Uno de los proyectos más reconocidos es el llamado Smart Dust (03) (*polvo inteligente*). Este concepto se introdujo en 2001 por Kristopher Pister de la Universidad de California. Nos encontramos ante una red inalámbrica de sensores o robots extremadamente pequeños (en torno a $1mm^3$), que pueden detectar señales de luz, vibraciones, temperatura, etc. Se trabaja para reducir el tamaño de estos sensores hasta alcanzar el mismo volumen que el de un grano de arena o una mota de polvo (de ahí el nombre *mote* para denominar a estos dispositivos). Los robots, compuestos por sensores, circuitos computadores, tecnologías de comunicación inalámbricas bidireccionales y una fuente de alimentación, forman, cuando están cerca, redes altamente flexibles y de baja potencia de forma automática.

En agosto de 2001 se implantó la red de sensores más grande conocida hasta el momento. Fue puesta en servicio durante un periodo corto de tiempo por la universidad de Berkeley y estaba compuesta por 800 nodos, y cuyo único fin era demostrar la potencia y escalabilidad de este tipo de redes.

Actualmente, las redes de sensores son un tema de investigación muy activo en universidades y ya hay aplicaciones comerciales basadas en estas redes. Incluso la NASA está integrando estos sistemas en sus proyectos de exploración en Marte con el fin de recabar la máxima información posible, y *Sun Microsystems, Inc.* está desarrollando un proyecto propio, el Sun Spot (04), basado en las redes de sensores.

2.1.2 Características de las WSN

A continuación, se describen las principales características de las redes de sensores.

2.1.2.1 Topología

Las redes de sensores son redes distribuidas, ya que los nodos cooperan entre sí para realizar una determinada tarea. Los nodos deben tener la capacidad de adaptación ante nuevas condiciones de red (e.g. entrada y/o salida de nodos en el vecindario) para poder comunicar la información que recogen del entorno.

Habitualmente se tiende a una computación cada vez más distribuida, donde los nodos sensores se comunican sólo con otros nodos sensores dentro de su vecindario, es decir, aquellos nodos que se encuentran dentro del alcance de su radio. Los nodos cooperan y ejecutan algoritmos distribuidos para conseguir una única respuesta global que el gateway se encargará de comunicar a la estación base.

2.1.2.2 Infraestructura

Las redes de sensores inalámbricas son redes *ad-hoc*, es decir, no necesitan una infraestructura de red definida, ya que sus nodos tienen la capacidad de poder actuar tanto como emisores, como receptores y como enrutadores de los datos. Cabe destacar la presencia de un nodo gateway o punto de acceso, que gestiona las comunicaciones con el resto de nodos y recolecta la información que se transmite a la estación base. En la Sección 2.2 se describirá en detalle la arquitectura de una red de sensores.

2.1.2.3 Consumo Energético

Los nodos, al tener un tamaño reducido, cuentan con una pequeña fuente de alimentación, típicamente formada por un par de baterías convencionales que alimentan la circuitería del nodo sensor y permiten su operación. Esto quiere decir que las limitaciones en cuanto a la duración de las baterías pueden ser críticas dependiendo de la finalidad de la red en uso. El tiempo de vida de un nodo sensor es por tanto el tiempo de la duración de sus baterías. Por este motivo los procesadores han de ser de bajo consumo, al igual que la radio, y se deben implementar soluciones software que optimicen el consumo energético. Se precisa que la duración de las baterías sea elevada, ya que en caso de que alguno de los motes las consuma completamente puede implicar un cambio importante en la topología, lo que implicaría un nuevo enrutamiento y organización de la red.

2.1.2.4 Costes de Producción

Estas redes fueron concebidas para estar compuestas por un número elevado de dispositivos, por lo que deben ser económicos. El modo de reducir el coste de la producción es su fabricación en grandes cantidades.

2.1.2.5 Tolerancia a Fallos

Los dispositivos que formen parte de la red de sensores han de ser capaces de funcionar pese a la existencia de fallos en su sistema o en la propia red. El fallo de un nodo de la red nunca debería implicar el fallo de toda la red. Todos los nodos han de tener la capacidad de autoconfiguración en caso de cambios en la topología de la red y de poder operar de manera desatendida a lo largo de su tiempo de vida.

2.1.2.6 Instalación

Las WSN son fáciles de implantar en cualquier entorno. En el caso de edificios no es necesario que se realicen obras para instalar los nodos ya que se trata de redes inalámbricas, y por tanto, no es necesario realizar ningún tipo de cableado. En el caso de espacios abiertos, tampoco se precisa de ningún tipo de instalación, sólo es necesario colocar los nodos en los puntos que se consideren óptimos para la toma de datos, ya que los propios nodos se auto-configuran y generan la red.

2.1.2.7 Variabilidad del Canal

El canal radio, que es el usado para la comunicación inalámbrica entre los motes, es un canal variable y que se ve afectado por problemas como atenuaciones, pérdidas de señal e interferencias, de modo que se pueden producir errores en los datos. En las WSN es característico el uso de protocolos tales como DSDV (**D**estination-**S**equenced **D**istance **V**ector routing), EWMA (**E**xponential **W**eighted **M**oving **A**verage) y AODV (**A**d-hoc **O**n-demand **D**istance **V**ector routing), aunque también está muy extendido el uso de broadcast (transmitir a todos los nodos de la red).

2.1.2.8 Tiempo Real

Muchas de las aplicaciones de WSN están dedicadas a los sistemas de seguridad y de prevención de catástrofes, como por ejemplo, guiar a los bomberos en el escenario de un incendio. Esto quiere decir que se necesitan mecanismos para la detección y actuación en tiempo real. Es preciso que en caso de detección de un determinado evento calificado como crítico, la información se emita inmediatamente a la estación base, donde un experto tomará las decisiones oportunas.

2.1.2.9 Heterogeneidad

Las WSN están formadas por distintos tipos de dispositivos, cada uno de ellos integrado por distintos componentes físicos (microcontrolador, radio, etc.), por lo que es necesaria la interoperabilidad del sistema. Tanto la comunicación como la configuración de los dispositivos se deben abstraer de las características hardware, con el fin de formar un sistema eficiente de alto nivel.

2.1.2.10 Seguridad

Las WSN utilizan el vacío como canal de transmisión, por lo que la información es muy vulnerable a ataques malintencionados. Se deben implantar mecanismos de integridad de datos, disponibilidad y confidencialidad específicos para estas redes, ya que los métodos tradicionales de seguridad no están adaptados.

2.1.2.11 Limitaciones de Hardware

Los nodos sensores son dispositivos con capacidades muy limitadas en cuanto a procesamiento, memoria, comunicación y, sobre todo, energía. Es necesario simplificar lo máximo posible el hardware con el fin de conseguir un consumo energético ajustado que permita maximizar el tiempo de vida de un nodo. También se precisan estas mismas características para el transceptor radio.

2.2 Arquitectura de una WSN

Una red inalámbrica de sensores consta de tres elementos principales: los nodos sensores, el gateway y la estación base, cada uno con diferentes funciones dentro de la red. En la siguiente figura se muestra la arquitectura básica de una WSN:

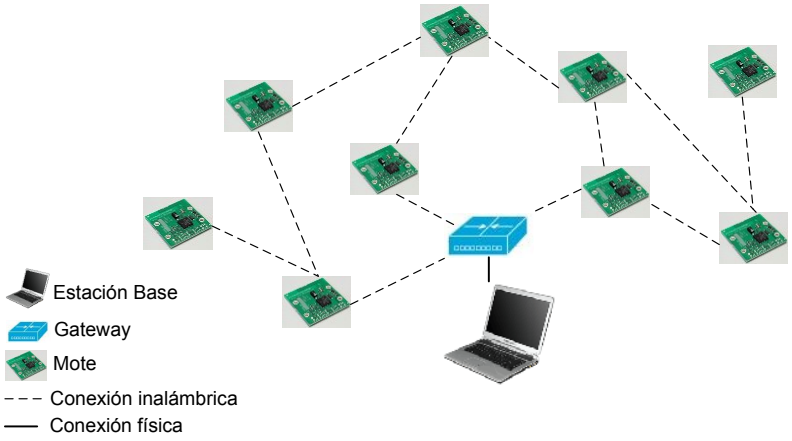


Ilustración 1: Arquitectura de una WSN

Detallaremos a continuación los elementos de hardware que componen una WSN.

2.2.1 Nodo Sensor

Un nodo sensor es un elemento computacional compuesto por un microprocesador con memoria, una interfaz de comunicación radio de baja potencia, uno o varios sensores y una batería. Mediante sensores o transductores toman la información del medio en el que se encuentran instalados y la convierten en señales digitales que se transmiten por la red.

Cada nodo puede ser dotado de varios sensores, por ejemplo, detectores de luz, temperatura, sensores de humedad, de velocidad, presión, etc. La Figura 2 muestra distintos tipos de sensores que pueden ser integrados con nodos sensores.

Ilustración 2: Diferentes tipos de sensores añadidos a los nodos sensores

A continuación, vamos a detallar de modo breve algunos de los nodos sensores más populares que emplean las WSN:

2.2.1.1 BT Node3

BT Node3 (05) está formado por una radio Bluetooth, una radio de baja potencia y un microcontrolador. Lo fabrica ETH Zúrich (Suiza) y está diseñado para plataformas de demostración y creación de prototipos de investigación en telefonía móvil, redes móviles Ad-Hoc (MANET, **M**obile **A**d-Hoc **NET**works) y redes de sensores. Usa el mismo módulo de radio que el sensor Mica2.

Ilustración 3: Nodo BTnode3

2.2.1.2 EyesIFX

EyesIFX (06) está desarrollado por Infineon en el proyecto EYES de WSN. Su microprocesador lo produce Texas Instruments. La plataforma integra sensores de temperatura y de luz, y ofrece soporte software proporcionado por TinyOS de TU Berlín.

Ilustración 4: Nodo EyesIFX

2.2.1.3 Imote2

Imote2 (07) se fabrica por Crossbow Technology Inc. Monta la CPU de baja potencia XScale PXA271 y una radio IEEE 802.15.4. Su diseño es modular y ampliable con conectores para tarjetas de expansión tanto en el lado superior como en el inferior.

Ilustración 5: Nodo Imote2

2.2.1.4 Iris

Iris (08) Fabricado por Crossbow Technology Inc., está diseñado para sistemas embebidos. Monta una radio de 2.4GHz (IEEE 802.15.4) de baja potencia con capacidad de transmisión de 250Kbps. También incluye conectores de expansión para diferentes sensores (temperatura, luz, humedad, etc.).

Ilustración 6: Nodo Iris

2.2.1.5 Mica2

Mica2 (09) es fabricado por Crossbow Technology Inc., pertenece a la tercera generación de módulos para redes de sensores. Monta un transmisor/receptor multicanal (868/916MHz). Estos nodos tienen capacidad de actuar de router en las comunicaciones inalámbricas y también cuenta con conectores de expansión para diferentes sensores

Ilustración 7: Nodo Mica2

2.2.1.6 Mica2Dot

Mica2Dot (10) (Crossbow Technology Inc.) también pertenece a la tercera generación de nodos sensores. Tiene forma de moneda y su tamaño es el de una cuarta parte que el nodo Mica2. Integra un sensor de temperatura, LEDs y un monitor de batería. Destaca su módulo radio multicanal (868/916MHz, 433/315MHz).

Ilustración 8: Nodo Mica2Dot

2.2.1.7 MicaZ

MicaZ (11), también está fabricado por Crossbow Technology Inc, pertenece a la misma familia que los nodos Mica, Mica2 y Mica2Dot. Ofrece características similares a los nodos descritos anteriormente.

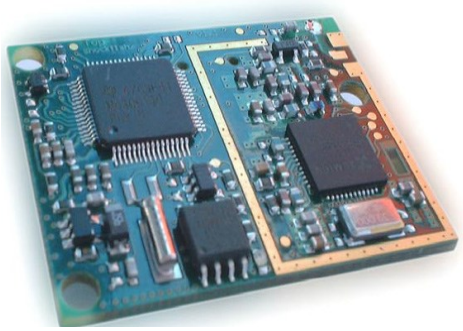
Ilustración 9: Nodo MicaZ

2.2.1.8 TelosB

TelosB (12), también llamado TPR2400 (Crossbow Technology Inc.), es una plataforma de investigación de código abierto. Incluye programación vía USB, radio IEEE 802.15.4 con un ratio de transmisión de 250Kbps, CPU de 2.4GHz con 10KB de RAM, bajo consumo y una flash de 1MB para almacenamiento de datos.

Ilustración 10: Nodo TelosB

2.2.1.9 TinyNode



Tinynode (13) es un módulo construido únicamente para ser usado con el sistema operativo TinyOS. Lo fabrica la empresa suiza Shockfish. Está diseñado para funcionar en potencias ultra bajas, e incorpora un sensor de temperatura, aunque la integración de otros tipos de sensores es posible. Usa un microprocesador MSP430 de Texas Instruments con radio multicanal de baja potencia.

Ilustración 11: Nodo Tinynode

2.2.1.10 Eko

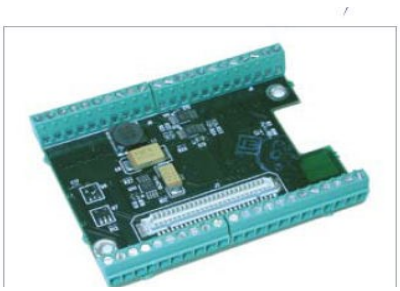
Eko (14) es un modelo de última generación, fabricado por Crossbow Technology Inc. Es un dispositivo muy robusto y diseñado para estar al aire libre y alimentarse por energía solar. Cada nodo puede tener hasta 4 sensores diferentes. Integra la plataforma Iris con baterías recargables y una célula solar. Los nodos están pre-programados y pre-configurados para formar una red y precisan entre 1 y 2 horas de exposición al sol al día para mantener sus baterías cargadas.

Ilustración 12: Nodo Eko

2.2.2 Placa de sensores

Las placas de sensores también son conocidas como placas de adquisición de datos. Estas placas integran transductores para realizar mediciones de luz, temperatura, humedad, etc. y se conectan con el nodo sensor a través de alguna interfaz de conexión. A continuación se definen algunas de las placas de sensores más comunes.

2.2.2.1 MDA300



MDA300 (15) es un modelo de placa de sensores fabricado por Crossbow Technology Inc. Integra un sensor de humedad y un sensor de temperatura y está diseñado para ser usado como una interfaz primaria entre los nodos IRIS y las familias Mica y Telos con los sensores externos. Es una placa que se puede usar para la monitorización del medio ambiente.

Ilustración 13: Placa de sensores MDA300

2.2.2.2 MDA320

La placa sensora MDA320 (16)(Crossbow Technology Inc) tiene hasta 8 canales de 16 bits para entrada de datos analógica. Es un dispositivo muy sencillo, que solo tiene entradas y salidas analógicas para poder interactuar con otros dispositivos. Esta placa de sensores es compatible con motes Mica2 y MicaZ

Ilustración 14: Placa de sensores MDA320

2.2.2.3 MDA100

MDA100 (17) es una placa de sensores fabricada por Crossbow Technology Inc, que integra sensores de temperatura y de luz Está diseñada para ser usada con motes IRIS, Mica2 Y MicaZ.

Ilustración 15: Placa de sensores MDA100

2.2.2.4 MTS300

La placa de sensores MTS300 (18) (Crossbow Technology Inc.) es una placa de sensores flexible, que integra sensores de luz temperatura y sonido. Esta placa puede usarse con motes IRIS, Mica2 Y MicaZ.

Ilustración 16: Placa de sensores MTS300

2.2.2.5 MTS310

MTS310 (19) está fabricada por Crossbow Technology Inc. Esta placa incluye un acelerómetro de doble eje, un magnetómetro de doble eje y sensores de luz, temperatura y sonido. Esta placa está diseñada para su uso con motes IRIS, Mica2 Y MicaZ.

Ilustración 17: Placa de sensores MTS310

2.2.2.6 MTS400

La placa MTS400 (20), fabricada por Crossbow Technology Inc., ofrece cinco sensores medioambientales, de modo que puede ser utilizado como una completa estación meteorológica, así como en entornos industriales, de agricultura, etc. Los sensores que incluye se caracterizan por ser eficientes energéticamente, lo cual alarga la vida útil de las baterías.

Ilustración 18: Placa de sensores MTS400

2.2.2.7 MTS420

Ilustración 19: Placa de sensores MTS420

La placa de sensores MTS420 (Crossbow Technology Inc.) es una evolución de la MTS400, a la que se le ha añadido un módulo GPS (además de los sensores de temperatura, humedad, luz, presión y acelerómetro). Esta placa está especialmente diseñada para su uso con motes MicaZ.

2.2.3 Gateway

El gateway o puerta de enlace conecta la red de sensores con la estación base, de modo que sirve de pasarela entre los protocolos de comunicación utilizados en ambos lados (ZigBee y TCP/IP, o un puerto serie del PC al que se encuentre físicamente conectado). Habitualmente lleva acoplado un nodo sensor (pero sin placa sensora) en el que se instala la aplicación a ejecutar por el gateway. Las prestaciones de este nodo suelen ser mejores que las de los nodos sensores al tratarse de un nodo crítico en el sistema. El gateway procesa todo el tráfico de la red, por lo que es sensible a que se produzcan cuellos de botella. Además, la alimentación de este nodo suele ser por red eléctrica, en lugar de depender de baterías.

A continuación, se describen los gateway de uso más común en las aplicaciones de las redes de sensores inalámbricas.

2.2.3.1 MIB510

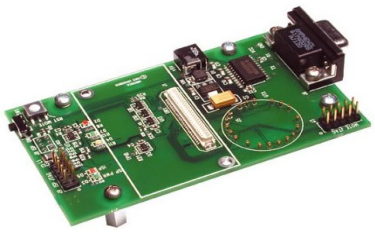


Ilustración 20: Gateway MIB510

MIB510 (21) es un módulo fabricado por Crossbow Tehnology Inc., permite que los datos de la red de sensores lleguen a la estación base. Es compatible con redes de sensores que utilicen plataformas Iris, MicaZ y Mica2. Además, proporciona una interfaz RS-232 de programación de nodos sensores.

2.2.3.2 MIB520

MIB520 (22) es un módulo también fabricado por Crossbow Technology Inc., compatible con las plataformas Iris, Mica2 y MicaZ. Como complemento a sus características básicas, ofrece una interfaz de programación de nodos sensores vía USB.

Ilustración 21: Gateway MIB520

2.2.3.3 MIB600

MIB600 (23), fabricado por Crossbow Technology Inc., ofrece conectividad Ethernet (10/100Base-T) a las plataformas Iris, MicaZ y Mica2. Permite el acceso remoto a los datos de los sensores a través de TCP/IP. Este modelo se conecta directamente a una red 10 Base-T LAN, al igual que cualquier otro dispositivo de la red

Ilustración 22: Gateway MIB600

2.2.3.4 NetBridge

Su denominación completa es Netbridge Stargate (24), también fabricado por Crossbow Technology Inc. Permite conectar todos los nodos Crossbow a una red Ethernet ya existente. Está basado en el procesador Intel XScale IXP420 a 266MHz. Además, incluye una conexión de cable Ethernet y dos puertos USB 2.0. También está equipado con una memoria flash de 8MB, 32 MB de RAM y 2 2GB de disco de sistema.

Ilustración 23: Gateway NetBridge

2.2.4 Estación Base

La estación base recopila los datos que se han recogido en la WSN. Habitualmente se trata de un ordenador muy potente, o un dispositivo embebido con mayores capacidades, donde los datos enviados desde el gateway son almacenados y posteriormente procesados para obtener estadísticas, histórico de datos, toma de decisiones y notificación de eventos.

En la estación base típicamente se analizan, procesan y visualizan los datos tomados por la red, aunque también puede encargarse de enviar los datos a otros dispositivos externos o incluso comunicarse con la propia red de sensores para el envío de mandatos.

2.3 Componentes físicos de los nodos sensores

El siguiente diagrama muestra un esquema de los componentes físicos que integran un nodo sensor:

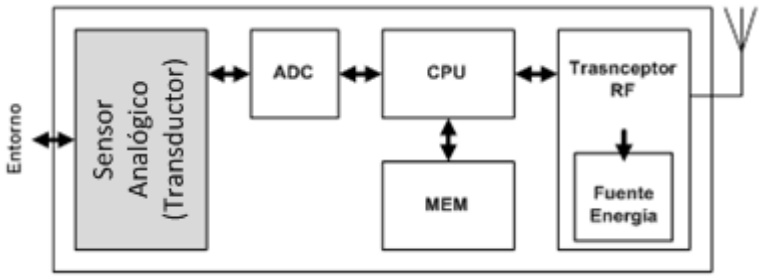


Ilustración 24: Estructura interna de un nodo sensor

Los componentes básicos de un nodo sensor son: la radio, el microcontrolador, una memoria (que puede ser interna o externa al micorprocesador), un conjunto de LEDs, y un concector de expansión para integrar una placa de sensores.

2.3.1 Radio

Una radio de un nodo sensor tiene típicamente una cobertura que oscila entre los 75 y los 100 metros en escenarios de exterior u *outdoor*, aunque esta cobertura es mucho menor en caso de escenarios de interior o *indoor*, dependiendo de la arquitectura del edificio. Se trabaja con distintas frecuencias, pero actualmente las más utilizadas son la banda de frecuencias de 2.4GHz (compatible con ZigBee) o de 933MHz.

La radio es el componente de hardware de mayor consumo energético. Debido a que el consumo energético es muy importante en los nodos sensores, las nuevas radios poseen distintos estados de más bajos consumos, con el fin de optimizar el consumo global del componente radio en el tiempo. Estos son los dos estándares más utilizados en el caso de las radios de los nodos sensores:

- **IEEE 802.15.4**: Es un estándar que define el nivel físico y el nivel de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos, propias de las redes Low-Rate Wireless Personal Area Network(LR-WPAN) (25). Define, por tanto, los dos niveles de red inferiores para dar servicio a un tipo específico de red inalámbrica de área personal, centrada en la habilitación de comunicaciones entre dispositivos ubicuos con bajo coste, velocidad y bajo consumo. Se enfatiza el bajo coste de la comunicación con nodos cercanos y sin infraestructura o con muy poca, para favorecer aún más el bajo consumo. Este estándar no define niveles superiores ni subcapas de interoperabilidad, razón principal por la que surgió la necesidad de la extensión conocida como Zigbee.
- **Zigbee** (26) es un protocolo abierto de comunicación usado principalmente en redes WPAN (Wireless Personal Area Network) y redes de sensores. Surge ante la necesidad de la existencia de un mecanismo de comunicación que sea muy eficiente en lo que al plano energético se refieren y además, sea seguro. Define los protocolos de red sobre el estándar IEEE 802.15.4. Principalmente se usa en radios digitales de bajo consumo, y funciona en frecuencias de 2.4GHz (esta frecuencia está liberada a nivel mundial). Casi todos los componentes de una red de sensores cuentan con una radio de este tipo. Gracias a estas características, los dispositivos cuentan con una mayor vida útil, puesto que apenas consume los recursos energéticos durante su funcionamiento. El tamaño del código es muy pequeño, por lo que también es muy adecuado en el caso de dispositivos con restricciones de hardware

2.3.2 Microcontrolador

El microcontrolador que integran los nodos sensores siguen típicamente una arquitectura Harvard, donde existe una memoria dedicada para el almacenamiento de datos en una memoria RAM (típicamente de 4KB) y una memoria dedicada al almacenamiento del programa en una memoria ROM (típicamente de 128KB). El microcontrolador trabaja en frecuencias bajas (normalmente entre 7 y 32MHz), y habitualmente su arquitectura es RISC de

8-16 bits. Incluye un conversor analógico–digital (ADC) y varios relojes o temporizadores para la sincronización de tareas.

2.3.3 Memoria

Existen varios tipos de memoria integradas dentro del microprocesador de los nodos o bien externas a él, las cuales se describen a continuación:

- **EEPROM**: Además de la memoria flash, poseen una memoria EEPROM en la que se guarda el programa que se esté ejecutando. Esta memoria se integra dentro del microcontrolador y típicamente, tiene un tamaño de 128 KB.
- **RAM**: Es el componente más restringido de las motes, ya que su tamaño no suele exceder de los 16KB. En esta memoria se almacenan los datos utilizados por el programa y del sistema operativo. Esta memoria reside dentro del microcontrolador. Alternativamente, cuando la aplicación requiere de mayor capacidad de almacenamiento, se puede usar la memoria flash como una extensión de la memoria RAM.
- **Flash**: En esta memoria los nodos almacenan los datos o las configuraciones de manera no volátil. Su tamaño va desde los pocos kilobytes a varios megabytes. El consumo energético de esta memoria es muy bajo y normalmente es externa al microcontrolador.

2.3.4 LEDs

Los nodos sensores cuentan con un mínimo de tres LEDs, normalmente de colores rojo, verde y amarillo. Estos LEDs son muy útiles en tiempo de desarrollo de la aplicación y a la hora de depurar la aplicación.

2.3.5 Conector de expansión

Los conectores de expansión son puertos de Entrada/Salida que permiten la interconexión de modo sencillo entre un nodo sensor y una placa en la que se incluyen varios sensores/actuadores. Cada nodo sensor (o cada familia de nodos) tiene un tipo de conector de expansión. Por ejemplo, el conector de la familia de motes Mica tiene 51 pines de conexión que permiten integrar la placa de sensores y el nodo sensor.

2.3.6 Fuente de alimentación

Típicamente, la fuente de alimentación más utilizada como baterías, son dos pilas alcalinas AA-LR6 de 1.5 voltios y de entre 2000 a 3000 mAh, que pueden durar desde cinco días hasta meses, dependiendo de la gestión más o menos eficiente que la aplicación haga de los componentes físicos. Existen otras fuentes de alimentación alternativas, tales como células solares, dinamos, etc., pero actualmente son todavía escasas las plataformas comerciales que las utilizan. Un ejemplo de este tipo de plataformas es Eko, descrita en este capítulo.

2.4 Despliegue de la red

Las aplicaciones WSN pueden desplegarse usando distintas topologías, tales como árbol, malla, estrella, o una híbrida de estas dos últimas. Cada topología cuenta con una serie de

ventajas e inconvenientes que se describen a continuación, pero antes debemos describir la clasificación de los nodos de la red en función del rol que asumen:

- **Nodos Finales:** Nodos sensores y actuadores donde se capturan los datos de los sensores. También se llaman *Reduced Functional Devices* (RFD).
- **Nodos Routers:** Nodos que permiten dar cobertura a redes muy extensas, pudiendo salvar obstáculos, problemas de congestión en la emisión de la información y posibles fallos en alguno de los componentes. También reciben el nombre de *Full Functional Devices* (FFD).
- **Nodos Gateway:** También llamados puertas de enlace, recogen los datos de la red y los reenvían a un dispositivo externo, sirviendo de punto de unión con una red LAN o Internet.

La topología se refiere a la configuración de los componentes hardware de la red, y cómo los datos son transmitidos a través de esa configuración. Cada topología es adecuada bajo ciertas circunstancias, y puede no ser la apropiada en otras. La siguiente figura representa el código de colores utilizado para distinguir cada uno de los nodos anteriores.

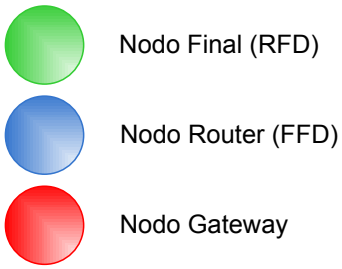


Ilustración 25: Representación gráfica de los distintos nodos de una WSN

2.4.1 Estrella

También conocida como *monosalto*, es una topología en la que la información enviada se envía en una única transmisión (salto) desde el origen al destino. Por tanto, todos los nodos han de estar en comunicación directa con el gateway (habitualmente a distancias entre los 30 y los 100 metros). Todos los nodos sensores son iguales, y el gateway capta la información de todos ellos. El gateway también es utilizado para emitir los datos al exterior y permite la monitorización de la red. Los nodos finales no intercambian información entre ellos, sino que usan el gateway para ello en caso necesario. La siguiente figura representa una topología en estrella.

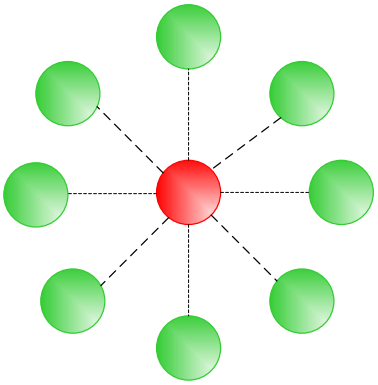


Ilustración 26: Representación de una red con topología en estrella

Esta topología es la que menor gasto energético desarrolla, pero está limitada por la distancia de transmisión vía radio entre cada nodo y el gateway. Tampoco existe un camino de comunicación alternativo en caso de que uno de los nodos no tenga visibilidad directa, lo que implica que la información de ese nodo se pierda.

2.4.2 Malla

Se trata de un sistema *multisalto*, donde todos los nodos son routers idénticos, y la información enviada por el nodo emisor puede tomar un número de saltos indeterminado hasta llegar al destino. Cada nodo puede enviar y recibir datos de cualquier nodo de la red y de gateway (la comunicación entre nodos es bidireccional). El tamaño de la red podrá ser, teóricamente, infinito, ya que la propagación de la información a través de los nodos hacia el gateway es siempre posible. Además, este tipo de red es altamente tolerante a fallos ya que no existe un camino único de comunicación con el gateway. Si uno de los nodos falla, la red se configurará alrededor del nodo caído automáticamente.

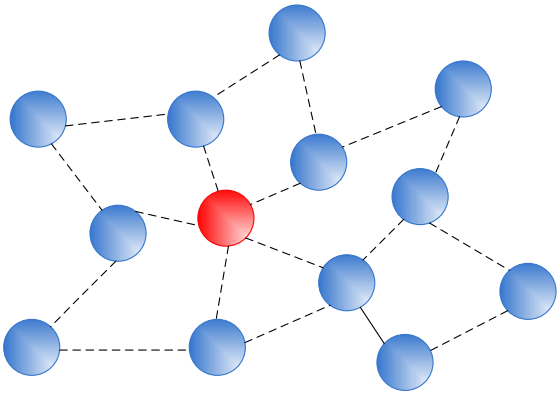


Ilustración 27: Representación de una red con topología en malla

Se pueden experimentar altos periodos de latencia a la hora de recibir la información, dependiendo del número de nodos y de la distancia que los separa.

2.4.3 Árbol

En la topología en forma de árbol, cada nodo se conecta a un nodo de jerarquía superior en el árbol hasta llegar al gateway. Los datos se encaminan desde el emisor a nodos de mayor jerarquía hasta alcanzar el gateway. Es la topología más eficiente de entre las anteriormente descritas, ya que los datos recorren el camino más eficaz (seleccionado vía algún algoritmo de enrutamiento) hasta llegar a la puerta de enlace. Por otro lado, no dispone de caminos alternativos, por lo que en caso de caída de algún nodo de la red, todos los nodos que se encuentren por debajo de la jerarquía quedarían aislados.

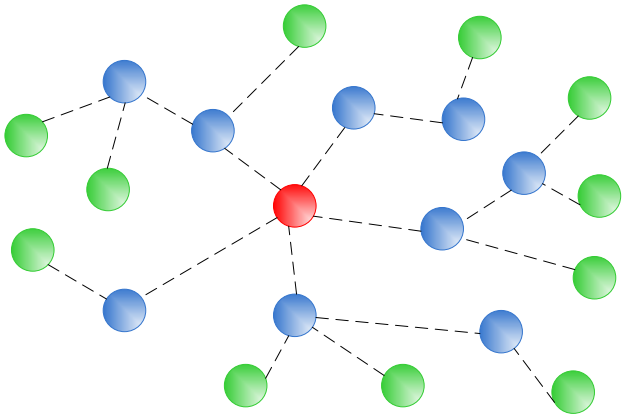


Ilustración 28: Representación de una red con topología en árbol

2.4.4 Híbrida Estrella-Malla

Esta topología busca combinar las ventajas de la topología en malla y en estrella, es decir, la simplicidad y bajo consumo de la topología en estrella con la posibilidad de cubrir una amplia extensión y reorganización de la red en caso de fallo que ofrece la topología en malla. Se crea una red en estrella alrededor de una red mallada. Los routers posibilitan la ampliación de la red y la corrección de fallos en estos nodos, y los nodos finales se conectan con los routers cercanos ahorrando energía.

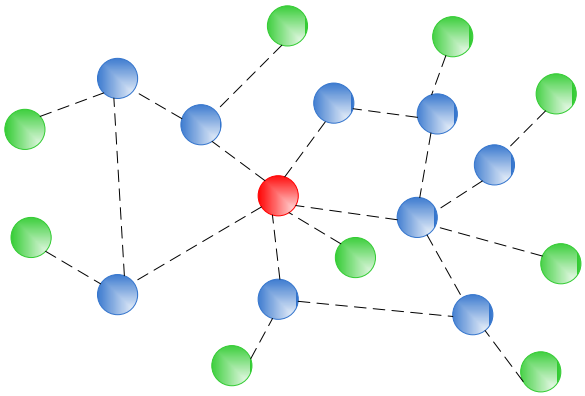


Ilustración 29: Representación de una red con topología híbrida estrella-malla

2.5 Sistemas Operativos para nodos sensores

En esta sección describimos los sistemas operativos diseñados específicamente para nodos sensores. Dadas las restricciones de cómputo, almacenamiento y energía de los nodos sensores, los sistemas operativos existentes para otros sistemas empotrados no pueden ser utilizados en este tipo de dispositivos. La función principal de estos sistemas operativos es gestionar eficientemente los recursos de hardware tan limitados y ofrecer una interfaz de programación sencilla a las aplicaciones.

2.5.1 TinyOS

TinyOS (27) es un sistema operativo de código abierto, diseñado específicamente para dispositivos sensores, por la universidad de Berkeley de California, Estados Unidos. Está desarrollado para que sea compatible con el mayor número de plataformas de sensores posible, pudiendo ser usado, además, en gran variedad de sistemas empotrados.

Ilustración 30 Logotipo de TinyOS

TinyOS está diseñado específicamente para nodos sensores, lo que proporciona optimizaciones con respecto a otros sistemas operativos, ajustándose a las necesidades y características de éstos. Entre las mejoras, destaca el reducido tamaño que ocupa en memoria RAM y ROM (*footprint*), el bajo consumo de energía, la capacidad de realizar operaciones de concurrencia, su gran diversidad en diseños y usos, y la posibilidad de generar operaciones robustas. TinyOS es un sistema operativo basado en eventos. Dispone de un kernel de pequeño tamaño, basado en una estructura de dos niveles: eventos (pensados para realizar procesos pequeños asíncronamente, que tienen la capacidad de interrumpir la tarea que se esté ejecutando) y tareas (pensadas para realizar un mayor procesamiento con la característica de no ser críticas en el tiempo, es decir, las tareas se ejecutan de manera asíncrona). Cuando un programa requiere la ejecución de una tarea, ésta se almacena en una cola de 8 posiciones (en la versión TinyOS 1.x) y 256 posiciones (en la versión 2.x). El comportamiento del planificador es el siguiente: cuando un evento es señalizado por el hardware (por ejemplo, la recepción de datos vía radio), el planificador ejecuta el manejador de eventos asociado. Si no hay evento que tratar, el planificador obtiene la primera tarea de la cola y la ejecuta. Cuando una tarea comienza, su ejecución se realiza en su totalidad y sólo puede ser interrumpida por un evento, es decir, no puede nunca ser interrumpida por otras tareas. TinyOS está escrito en nesC(28), un metalenguaje derivado de C, diseñado para responder a las necesidades que existen en los sistemas embebidos. El modelo de programación está orientado a componentes. Esto quiere decir que, una aplicación escrita en nesC, puede ser vista como un conjunto de componentes relacionados entre sí. Se basa en los siguientes conceptos:

- Separación entre la composición y la implementación de una aplicación. Los programas se construyen a partir de componentes, que se ensamblan para formar programas complejos.
- El comportamiento de los componentes viene dado por sus interfaces. Un componente puede usar y/o proveer una o varias interfaces y las interfaces que pretenden representar la funcionalidad del componente que las provee. Las interfaces usadas representan la funcionalidad que el componente necesita para realizar su trabajo.
- Las interfaces son bidireccionales, cada interfaz especifica un conjunto de funciones que son implementadas por el proveedor y a su vez determina un conjunto de funciones que deben ser implementadas por el componente que usa la interfaz.

Normalmente, los comandos realizan una llamada hacia un nivel de abstracción inferior, es decir, de componentes de aplicación a componentes más próximos al hardware, mientras que los eventos realizan la llamada hacia un nivel de abstracción superior.

2.5.2 Contiki

Contiki (29) es un sistema operativo de libre distribución, de código abierto. Está desarrollado para su uso en sistemas empotrados, desde pequeños microcontroladores de 8 bits a sistemas empotrados de mayor capacidad. El núcleo del sistema operativo y la mayoría de las funciones principales fueron desarrollados por Adam Dunkels en el grupo de sistemas de redes empotradas en el instituto sueco de ciencias computacionales (SICS). Contiki consiste en un núcleo orientado a eventos, que hace uso de *threads* (hilos) muy ligeros denominados *protothreads* (30), sobre los que los programas son cargados y descargados dinámicamente. Un proceso Contiki es un protothread y un protothread puede ser visto como un manejador de eventos bloqueante. El modelo de programación de Contiki usa una sintaxis bloqueante y todos los protothread comparten la misma pila. Además, soporta multi-threading, comunicación entre procesos mediante paso de mensajes a través de eventos, además de un subsistema GUI opcional que pueden usar un soporte directo de gráficos para terminales locales, terminales virtuales en red mediante VNC o sobre Telnet. Contiki soporta multithread como librería; a pesar de ser multitarea e incluir soporte a una pila TCP/IP muy ligera, sólo requiere varios KB de código y unos cientos de RAM. Este footprint, sin embargo, es superior al requerido por TinyOS.

Contiki soporta una gran variedad de plataformas de nodo sensor y otros dispositivos empotrados, desde microcontroladores empotrados como el MSP430 y el AVR, a viejos ordenadores domésticos.

2.5.3 MANTIS

MANTIS (MultimodAl NeTworks In-situ Sensors) (31) es un sistema operativo para nodos sensores desarrollado en la Universidad de Colorado en 2003. Está escrito en C, y es de código abierto basado en POSIX (Portable Operating System Interface). Mantis soporta multi-threading con un planificador basado en prioridades. Ofrece un gran abanico de posibilidades de configuración, por ejemplo:

- Configuración de la transición de estados de dispositivos (abierto, cerrado, etc.).
- Se puede establecer un número indeterminado de estados para cada tarea.
- Se pueden introducir diferentes perfiles de usuario (programador, tester, etc.).

Una de las ventajas principales es que los procesos largos no bloquean el sistema, y además, su modelo de programación, basado en POSIX, es más sencillo que el basado en eventos. Por el contrario, es sensible a la sobrecarga debida a los cambios de contexto y tiene un mayor consumo de memoria debido a la existencia de múltiples pilas (una por proceso).

2.5.4 LiteOS

LiteOS (32) es un sistema operativo diseñado en principio para su uso en calculadoras, pero también se ha utilizado para redes de sensores. También se trata de un sistema multithread, como MANTIS.

2.6 Tecnologías de Software

A continuación, se describen algunas de las tecnologías de software utilizadas en el desarrollo de este proyecto.

2.6.1 Java

El lenguaje de programación Java (33) fue desarrollado por James Gosling de Sun Microsystems a principios de los años 90, y publicado en 1995 como un componente fundamental de la plataforma de Java. Se trata de un lenguaje de programación orientado a objetos. Surge como evolución de los lenguajes C y C++, pero tiene menos facilidades de bajo nivel que cualquiera de ellos. La compilación de los programas desarrollados en este lenguaje se hace, generalmente, en un bytecode (clase Java). Para que los bytecodes puedan ejecutarse, necesitan utilizar la máquina virtual de Java JVM, por lo que cualquier dispositivo que la tenga instalada podrá ejecutar cualquier aplicación. Esto es posible gracias a que Java proporciona una capa de abstracción sobre el hardware en el que se ejecuta, de manera que la ejecución de un mismo código es posible en máquinas de diferente índole.

Java es un lenguaje de programación de propósito general, concurrente, basado en clases y orientado a objetos diseñado principalmente para tener tan pocas dependencias de implementación como fuera posible. Java se creó con cinco objetivos principales:

- Orientación a objetos: la metodología de la programación orientada a objetos es la que se debería utilizar.
- Portabilidad: debe permitir la ejecución de un mismo programa en varios sistemas operativos.
- Debe incluir, por defecto, soporte para el trabajo en red.
- Debe poder ejecutar código en sistemas remotos de forma segura.
- Debe ser sencillo de utilizar y tomar lo mejor de otros lenguajes orientados a objetos, como por ejemplo, C++.

Las características principales que definen este lenguaje de programación son:

- Orientado a objetos: La orientación a objetos se refiere a un método de programación y al diseño del lenguaje. Principalmente, la idea es diseñar el software de modo que los datos que se usen estén unidos a sus operaciones. De este modo, los datos y el código se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el “comportamiento” (código) y el “estado” (datos). El principio es separar aquello que cambia de las cosas de las que permanecen inalterables. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema software. Se pretende que los grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. También facilita la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos. Así, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software a construir proyectos de envergadura empleando componentes ya existentes y de calidad. La reutilización del software ha experimentado resultados dispares, encontrando dos dificultades principales: el diseño de objetos es pobremente comprendido, y la falta de una metodología para la amplia comunicación de oportunidades de reutilización.

- Independencia de la plataforma: Esto significa que los programas escritos en Java pueden ejecutarse en cualquier tipo de hardware, cumpliendo el axioma de Java "write once, run everywhere" (escriba el programa una vez, y ejecútelo en cualquier dispositivo) El código fuente escrito en Java se compila y genera un código "bytecode" que está a medio camino entre el código fuente y el código máquina. Este bytecode se ejecuta en la JVM (máquina virtual), programa que interpreta y ejecuta el código. También se suministran bibliotecas adicionales para acceder a las características de cada dispositivo.
- Recolector de basuras. El problema en Java de fugas de memoria se evita, en gran medida, gracias a la recolección de basuras (automatic garbage collector). El propio programa determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java (Java runtime) es el responsable de la gestión del ciclo de vida de los objetos. El programa o algunos objetos pueden tener localizado un objeto mediante una referencia a éste. Cuando no quedan referencias a un objeto, el recolector de basuras de Java lo borra, liberando así la memoria que ocupaba y previniendo posibles figas. Aun así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios. El recolector de basuras de Java permite una fácil creación y eliminación de objetos y una mayor seguridad.

2.6.2 Simulador Avrora

Avrora (34) es un simulador para aplicaciones de redes de sensores basadas en TinyOS. Se trata de un proyecto desarrollado en la Universidad de California (Los Ángeles). Está formado por un conjunto de herramientas diseñadas para la simulación y análisis de programas implementados para los microcontroladores de Atmel (Atmega 128, Atmega 32 y Atmega16), que son los microcontroladores que usan las motes de la familia Mica e Iris, por lo que Avrora se puede usar con programas diseñados para cualquiera de ellas. El lenguaje de programación en el que Avrora está desarrollado es Java, por lo que es fácil de ejecutar en casi cualquier entorno, siendo sólo necesaria la instalación de la máquina virtual de Java JVM. Avrora dispone de un conjunto de monitores que permiten analizar los recursos de la aplicación simulada, tales como energía consumida por los componentes, números de ciclos de CPU usados por cada uno de ellos o el tiempo en el que cada componente ha estado en un determinado estado. También se puede analizar la memoria consumida por la aplicación, simular varias aplicaciones a la vez sobre la misma red de sensores inalámbrica, incluso simular la aplicación en tiempo real.

Avrora es un proyecto de código abierto por lo que además puede extenderse fácilmente e incluir nuevas funcionalidades. La implementación y la integración de nuevas funcionalidades es bastante sencilla, ya que la estructura de Avrora está diseñada para este fin.

2.7 Aplicaciones de las WSN

Actualmente, existe un conjunto amplio de aplicaciones donde las WSN podrían incrementar el conocimiento que los seres humanos tenemos sobre el entorno. La información proporcionada por estas redes puede ser utilizada por los científicos para tomar decisiones que permitan incrementar nuestra calidad de vida. A continuación, enumeramos algunas de estas aplicaciones.

2.7.1 Eficiencia Energética

En la actualidad, las nuevas legislaciones de protección y sostenibilidad, el incremento del precio de la energía y una mayor y creciente conciencia medioambiental, impulsan en gran medida el uso racional de la energía, no solo por parte de usuarios finales, sino también de grandes consumidores, y con ello, los sistemas de mejora de la eficiencia energética en edificios.

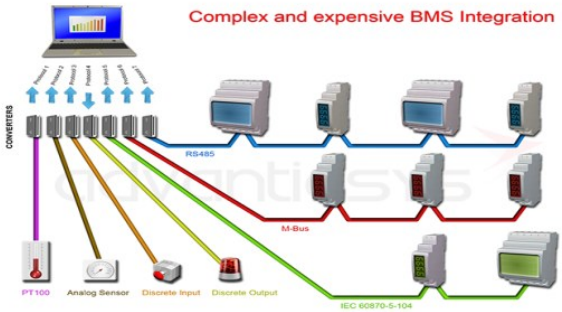


Ilustración 31 Solución de mejora energética

En estos casos, las redes de sensores se utilizan para controlar el uso eficaz de la electricidad y más valores como la iluminación, la climatización, consumos de agua, etc. La cadena de supermercados *Máxima* ha implementado esta solución en todas sus sedes, logrando un ahorro medio tras la implantación del sistema que oscila entre el 10% y el 15% en cada tienda (35).

2.7.2 Monitorización de Cultivos

El uso de redes de sensores inalámbricas en agricultura es cada vez más común, ya que ofrece múltiples beneficios, tales como la mejora de la calidad del cultivo y la reducción de los costes operativos. Las redes de sensores solucionan los problemas de monitorización en exteriores ya que no son necesarios ni cables ni fuentes de alimentación en lugares con carencias de infraestructuras. El despliegue de estas redes facilita que las muestras recogidas sean numerosas y consistentes. La empresa Advanticsys (36) ha desarrollado todos estos campos:

- Agricultura de precisión:

Ilustración 32: Agricultura de precisión

Las redes de sensores inalámbricas permiten realizar un seguimiento preciso del desarrollo de los cultivos. Ésto permite conocer el estado en el que se encuentran los productos en cada momento, facilitándose saber la fecha óptima de cosecha (gracias a la medición del diámetro del tronco, tallo y de los frutos, la implantación de estaciones meteorológicas locales con anemómetros, pluviómetros y medidores del viento tanto del momento como las esperadas). Se consigue que los nodos sensores tengan una autonomía extendida, ya que se pueden usar tanto baterías como fuentes de alimentación alternativas que permitan la recarga de las mismas, aprovechando el sol, viento, etc.

- Riego inteligente:

Puede lograrse una mayor eficiencia de los sistemas de riego si se tienen los datos de humedad y temperatura del suelo en tiempo real. Si estos conocimientos se combinan con un sistema individualizado de irrigación, cada planta recibe la cantidad de agua precisa para su crecimiento, de modo que se reducen los costes y se mejora la calidad de los cultivos. Incluso en estos casos, los elementos de la WSN se pueden usar para controlar los elementos del sistema de riego, sin necesidad de una estructura cableada compleja.

Ilustración 33: Riego inteligente

- Invernaderos:

Ilustración 34: Invernaderos

En el caso de los cultivos en invernaderos, donde se precisan unas condiciones microclimáticas muy concretas en cada momento, las redes de sensores facilitan las mediciones tanto de humedad, como de temperatura en diferentes puntos. Se han comenzado a implantar en grandes explotaciones y para cultivos muy específicos, por lo que es necesario que los sensores sean equipos muy robustos, con poca probabilidad de fallo, y que, por tanto, las baterías tengan una gran autonomía.

2.7.3 Entornos Militares

Dado que el origen de las WSN está dentro del campo de la seguridad militar, las aplicaciones dentro de este campo son unas de las más avanzadas. Se han desplegado redes para monitorizar equipos, zonas estratégicas, detectar ataques biológicos, químicos e incluso nucleares. El uso de redes de sensores para la detección de submarinos (37) se puede considerar como el origen del estudio y evolución de las WSN. En la imagen se puede observar el funcionamiento para esta aplicación.

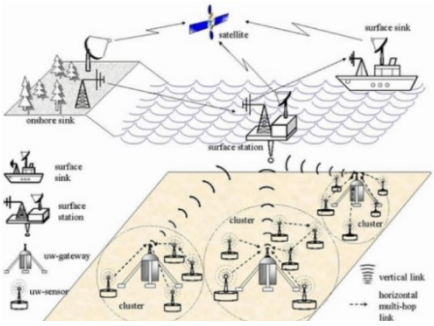


Ilustración 35: Detección de submarinos

2.7.4 Automoción

Las WSN pueden informar de la situación del tráfico en ángulos muertos que las cámaras fijas de tráfico no cubren. También pueden avisar a los conductores en caso de atasco o accidente en la carretera que estén transitando o a la que se dirijan, dándoles la capacidad de elección de una ruta alternativa. Los vehículos portadores de sensores pueden formar una gran red. Los nodos transmitirán la información relativa a la situación, velocidad, dirección, etc., que permitirá poder realizar un control de tráfico en diferentes situaciones. La mejora de la seguridad y la eficiencia del transporte es el objetivo de Trackass (**T**echnologies for **R**oad **A**dvanced **C**ooperative **K**nowledge **S**haring **S**ensors) (38).

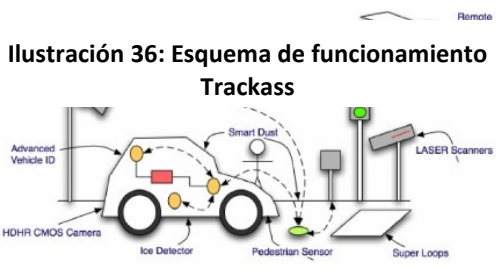


Ilustración 36: Esquema de funcionamiento Trackass

2.7.5 Estudios Medioambientales

El estudio de la evolución de variables físicas como la humedad, temperatura o la actividad sísmica permiten el análisis de problemas y prevención de desastres medioambientales (39). También son útiles para la detección de plagas. Gracias al gran número de sensores desplegados en un área geográfica amplia se puede localizar individuos infestados y aislarlos rápidamente con el fin de minimizar la expansión de la enfermedad. También se están implantando en explotaciones petrolíferas, para medir la presión del petróleo y prevenir de este modo los problemas que puedan existir en la extracción del crudo.

Ilustración 37: Medición de la presión en pozo petrolífero

2.7.6 Domótica

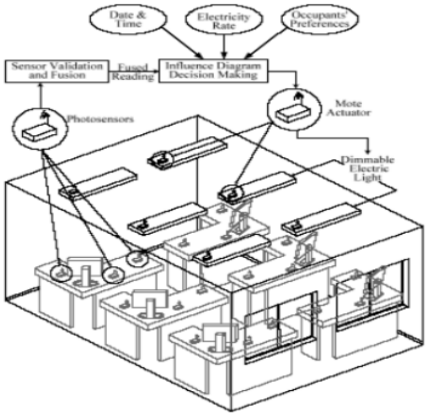


Ilustración 38: Uso de sensores en hogares

Este tipo de redes, de sencilla implantación debido a su reducido tamaño, hacen que sea una tecnología ideal para domotizar el hogar a un precio asequible. Se consigue controlar la iluminación, la temperatura y la detección de presencia tanto para favorecer el ahorro energético como para la captación de intrusos. Cada día es más habitual la instalación de sensores en casas, sobre todo, de reciente construcción, que hagan mediciones del gasto energético de las lámparas y aparatos electrónicos (40).

2.7.7 Aplicaciones Sociales y Médicas

El campo de la medicina y la acción social es otro en el que el desarrollo de las redes de sensores está siendo muy importante. En el caso de pacientes que requieren un control exhaustivo de sus niveles de azúcar, presión arterial, etc., la medición y monitorización de constantes en tiempo real mejora notablemente el tiempo empleado para la detección de anomalías de sus patologías y por tanto, disminuye el tiempo de respuesta en la atención urgente al paciente. También implican una mejora en la calidad de vida de los pacientes crónicos, ya que no es necesario que se desplacen a su hospital de referencia o centro de salud para realizarse los seguimientos de sus enfermedades. Los datos relevantes de la salud de los enfermos se toman en sus propias casas y se remiten a su médico. El seguimiento en estos casos es exhaustivo y el seguimiento del paciente se realiza en profundidad. Relacionado con este campo se encuentra el proyecto Code Blue (41), pensado para la asistencia médica de emergencia en caso de catástrofes naturales o ataques terroristas.

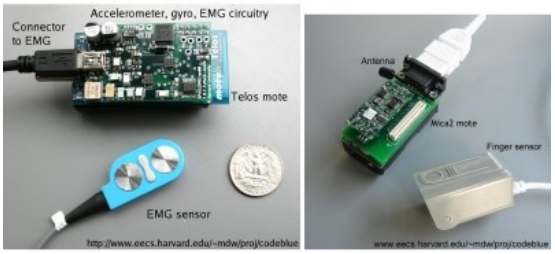


Ilustración 39: Sensores Code Blue

3. Especificación de requisitos

En este capítulo se va a presentar la especificación de requisitos del proyecto, para lo cual se van a definir los requisitos, particularidades y restricciones del problema. La especificación de requisitos desarrollada para este proyecto ha seguido la metodología Métrica (42), una metodología de planificación, desarrollo y mantenimiento de sistemas de información desarrollado por el Ministerio de Hacienda y Administraciones Públicas de España. A continuación se describen los requisitos funcionales, requisitos de apariencia, de rendimiento y del sistema.

En este proyecto se pretende desarrollar un módulo de Java que pueda integrarse con el simulador Avrora. Este nuevo módulo ofrece una interfaz gráfica para las simulaciones de aplicaciones de redes de sensores que se desarrollen utilizando Avrora, de manera que el usuario pueda visualizar de una manera sencilla los resultados de esa simulación y otros resultados adicionales que se calculan, como es el análisis del tráfico de paquetes y la topología de nodos de la red.

3.1 Requisitos Funcionales

En este apartado se detallan todos los requisitos funcionales necesarios en este proyecto.

Identificador: UR-F001	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	Se deberá desarrollar un módulo que permita la monitorización y visualización de los resultados de una simulación ejecutada con el simulador Avrora

Tabla 1 - Requisito funcional UR-F001

Identificador: UR-F002	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El nuevo módulo se deberá integrar con la herramienta de simulación de redes de sensores Avrora

Tabla 2 - Requisito funcional UR-F002

Identificador: UR-F003	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El módulo deberá monitorizar la simulación del tráfico de red que se genera en una red de sensores simulada con la herramienta Avrora.

Tabla 3 - Requisito funcional UR-F003

Identificador: UR-F004	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El módulo deberá analizar el tráfico de paquetes de la simulación. Los paquetes que pueden enviarse/recibirse por los nodos sensores son de los siguientes tipos: 1- Paquetes de control o *beacons:* gestión de la topología y red 2- Paquetes de datos: datos de la aplicación 3- Paquetes ACK: confirmación de la recepción de un paquete de datos

Tabla 4 - Requisito funcional UR-F004

Identificador: UR-F005	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	La red de sensores a simular deberá estar formada por un conjunto de N (N>0) nodos sensores y un nodo especial, el Gateway, que actúa como puente entre la red y un PC.

Tabla 5- Requisito funcional UR-F005

Identificador: UR-F006	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El nuevo módulo se integrará con el simulador Avrora vía un nuevo monitor denominado Network, que podrá ejecutarse de dos maneras diferentes: • en modo gráfico: se lanzará la GUI • en modo texto: no se lanzará la GUI

Tabla 6 - Requisito funcional UR-F006

Identificador: UR-F007	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	Los nodos sensores ejecutarán una aplicación desarrollada en TinyOS y compilada para los nodos sensores Mica2 o MicaZ. Esta aplicación permitirá el envío y transmisión de los mensajes identificados en el requisito UR-F004.

Tabla 7 - Requisito funcional UR-F007

Identificador: UR-F008	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El módulo a desarrollar deberá proveer una interfaz gráfica para presentar, en tiempo de ejecución de la simulación, al menos, la siguiente información: 1- El despliegue y la topología de la red: localización de los nodos sensores al principio y al finalizar la simulación. 2- La transmisión y recepción de un mensaje: nodo emisor y receptor. 3- El identificador de los nodos de la red. 4- El tráfico de red entre los nodos sensores. 5- Los resultados de energía consumida por cada nodo en la simulación.

Tabla 8 - Requisito funcional UR-F008

Identificador: UR-F009	
Prioridad: Alta	**Fuente:** Tutora
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	Durante la simulación, el sistema deberá identificar y monitorizar tanto los paquetes de datos como los paquetes de control de tráfico transmitidos entre los nodos.

Tabla 9 - Requisito funcional UR-F009

Identificador: UR-F010	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	Se representará gráficamente la transmisión y recepción de paquetes entre dos nodos (origen y destino) de la red.

Tabla 10 - Requisito funcional UR-F010

Identificador: UR-F011	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	Al término de una simulación el sistema deberá visualizar un informe resumen de la simulación

Tabla 11 - Requisito funcional UR-F011

Identificador: UR-F012	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	Los *logs* de ejecución de la simulación se almacenarán en un fichero de datos.

Tabla 12 - Requisito funcional UR-F012

Identificador: UR-F013	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	En cuanto al consumo energético de los nodos, se detallarán los siguientes parámetros: 1- El identificador del nodo 2- El consumo energético asociado a la CPU medido en Julios 3- El consumo energético de los LED (amarillo, verde, rojo) medido en Julios 4- El consumo energético del módulo de radio medido en Julios 5- El consumo energético de la memoria flash medido en Julios

Tabla 13 - Requisito funcional UR-F013

Identificador: UR-F014	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El informe resumen de la simulación se mostrará al finalizar una simulación e incluirá: 1- Tabla resumen con el número de paquetes que cada nodo envía a cada uno de los nodos de la red 2- Número de paquetes enviados por cada nodo de la red y por cada tipo 3- Número de paquetes recibidos por cada nodo de la red y por cada tipo (datos, beacons, ACKs) 4- Topología inicial (al comienzo de la simulación) y final (en el momento de finalización de la simulación) 5- Tabla resumen con el consumo energético de cada uno de los nodos que forman parte del sistema. Para cada nodo se mostrará el componente, estado y consumo (en Julios)

Tabla 14 - Requisito funcional UR-F014

3.2 Requisitos de apariencia

En este apartado se detallan los requisitos relacionados con la apariencia de la interfaz gráfica del sistema.

Identificador: UR-A001	
Prioridad: Media	**Fuente:** Tutor
Necesidad: Opcional	
Claridad: Media	**Verificabilidad:** Media
Descripción	Se usará una línea de color rojo entre el nodo emisor y el nodo receptor para identificar la ruta por la que se está produciendo un envío de paquetes entre dos nodos

Tabla 15 - Requisito de apariencia UR-A001

Identificador: UR-A002	
Prioridad: Media	**Fuente:** Tutora
Necesidad: Opcional	
Claridad: Media	**Verificabilidad:** Media
Descripción	La imagen del nodo Gateway será diferente a la del resto de nodos, de modo que sea más sencillo distinguirlo de los demás e identificar claramente que se encuentra conectado a la estación base

Tabla 16 - Requisito de apariencia UR-A002

3.3 Requisitos de rendimiento

A continuación se encuentran los requisitos relacionados con el rendimiento del sistema.

Identificador: UR-R001	
Prioridad: Media	**Fuente:** Tutor
Necesidad: Opcional	
Claridad: Alta	**Verificabilidad:** Media
Descripción	El sistema deberá poder mostrar la información de gran cantidad de nodos sin que el rendimiento global del sistema se vea afectado de manera significativa respecto al rendimiento de la propia plataforma Avrora

Tabla 17 - Requisito de rendimiento UR-R001

3.4 Requisitos de sistema

Los requisitos de sistema son aquellos que especifican el hardware o software relacionados con la aplicación.

Identificador: UR-S001	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El sistema se desarrollara sobre la plataforma Avrora versión 1.7.106.

Tabla 18 - Requisito de sistema UR-S001

Identificador: UR-S002	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Esencial	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El lenguaje de programación utilizado será Java versión 7, actualización 45 (compilación 1.7.0_45-b18)

Tabla 19 - Requisito de sistema UR-S002

Identificador: UR-S003	
Prioridad: Alta	**Fuente:** Tutor
Necesidad: Opcional	
Claridad: Alta	**Verificabilidad:** Alta
Descripción	El módulo será multiplataforma, al igual que Avrora, y deberá funcionar tanto en el sistema operativo Windows como Linux.

Tabla 20 - Requisito de sistema UR-S003

4. El simulador Avrora

En este capítulo se describirá el simulador Avrora para poder entender la solución implementada a lo largo de este proyecto.

Tal y como se ha explicado con anterioridad en el apartado *2.4.2 Simulador Avrora,* Avrora es un potente simulador para aplicaciones de redes de sensores implementadas en TinyOS, que fue desarrollado en la Universidad de California, con el que se puede simular y analizar el comportamiento de las aplicaciones ejecutadas sobre los microcontroladores de Atmel. Avrora permite simular aplicaciones sobre las motes de la familia Mica e Iris, anteriormente descritas en el capítulo 2. La simulación con Avrora del comportamiento de una red de sensores arroja los resultados en formato texto codificado en hexadecimal, por lo que la interpretación de estos datos es compleja. Este capítulo pretende presentar los parámetros de entrada del simulador y describir los resultados que proporciona Avrora en una simulación.

4.1 Parámetros de entrada

Para poder ejecutar la simulación de una red de sensores inalámbricos es necesario aportar una serie de parámetros de entrada en Avrora. Una simulación de Avrora usa un conjunto de parámetros de entrada. En particular, para el desarrollo de este PFC se han incluido los siguientes parámetros de entrada:

```
-action=simulate -simulation=sensor-network -nodecount=1,4 -platform=micaz
-input=elf -seconds=40 -report-seconds -monitors=energy,packet -radio-
range=75 -topology=static -topology-file=topology5nodes.out
TestNetworkLPLBS.elf TestNetworkLPLNodes_50dc_20.elf
```

Cada parámetro de entrada se especifica como un guión (-) seguido de un flag (nombre del parámetro) y un igual (=), tras el cual se dará la información que queremos que se use en la ejecución. Los parámetros incluidos son los siguientes:

- **-action,** indica la acción que se va a llevar a cabo en la ejecución. En nuestro caso, es "simulate", es decir, simulación de una aplicación.
- **-simulation** indica qué es lo que se va a simular durante la ejecución de Avrora. En nuestro caso, es "sensor-network", es decir, una red de sensores.
- **-nodecount** indica el número de nodos con los que se llevará a cabo la ejecución. El número de nodos se incluye en formato "x,y", donde "x" es el número de nodos gateway (y por tanto, siempre tomará el valor 1) e "y" es el número de nodos sensores.
- **-platform** indica el tipo de nodos sensores que se van a usar durante la simulación. En nuestro caso, los nodos sensores son "micaz".
- **-input** indica el formato que tienen los ficheros ejecutables que se usan para la ejecución. Existen diversos formatos como .EXE, .SREC o .ELF. El formato de ficheros en nuestro caso es "elf" .
- **-seconds** indica el tiempo en segundos durante el cual se estará ejecutando la aplicación de red de sensores.
- **-report-seconds** causa que todos los tiempos que imprime en pantalla el simulador se muestren en segundos, en lugar que en número de ciclos del reloj.

- **-monitors** indica los monitores de simulación que queremos utilizar. Al término de la simulación, cada monitor reporta la información calculada durante la ejecución. En nuestro caso, estamos interesados en conocer el consumo energético de cada nodo y los paquetes transmitidos y recibidos por cada nodo, por lo que se usan los monitores "energy" y "packet", respectivamente.
- **-radio-range** indica el alcance de la radio de los nodos sensores. En este caso, hemos establecido un rango de alcance de 75 metros, al suponer un escenario *outdoor.*
- **-topology** especifica el nombre de un fichero que contendrá la información de topología de la red de sensores. Para este comportamiento, esta opción debe ser configurada con el parámetro "static".
- **-topology-file** indica el nombre de los ficheros en los que está almacenada la topología que se tendrá en cuenta para la ejecución de la simulación de la red de sensores inalámbricos.

4.2 Análisis de los resultados de la simulación

Durante la simulación y al término de la misma Avrora mostrará los resultados en la consola de ejecución de Java. Todos los paquetes transmitidos y recibidos por cada nodo se muestran inmediatamente en la consola, dado que el monitor *packet* fue indicado en la entrada. Las primeras líneas resultado que tenemos son las siguientes:

```
[1;34mAvrora _[1;00m[_[1;34mBeta 1.7.113_[1;00m] - (c) 2003-2007 UCLA
Compilers Group
[0;33mLoading TestNetworkLPLBS.elf_[1;00m..._[0;32mOK_[1;00m
[0;33mLoading TestNetworkLPLNodes_50dc_20%.elf_[1;00m..._[0;32mOK_[1;00m
```

Inicialmente, se indica que nos encontramos ante Avrora, desarrollado por UCLA Compilers Group, y tras esto, nos muestra la carga de los ficheros ejecutables *TestNetworkLPLBS.elf y TestNetworkLPLNodes_50dc_20%.elf* para el gateway y para los nodos sensores, respectivamente. Tras estas tres líneas, se indica el comienzo de los eventos de simulación:

```
=={ Simulation events }=====================================================
```

A continuación vamos a analizar cada uno de los monitores que hemos solicitado a Avrora que nos muestre durante la simulación.

4.2.1 Monitor *packet*

Todos los paquetes que se intercambian durante la simulación tienen el mismo formato. Se usan tiras de bytes en formato hexadecimal para representar tanto las cabeceras como el cuerpo del paquete de datos. Primero se muestra el identificador de nodo, donde el nodo 0 (representado como 00) se corresponde con el nodo sensor que tiene la labor de gateway. A continuación, se muestra la hora en la que tiene lugar la transmisión/recepción del paquete, seguido de un indicador con flechas que indica si el mensaje es transmitido por el nodo (secuencia "---->") o si el mensaje es recibido por el nodo (secuencia "<====”). Tras esta información, se puede ver el contenido del paquete que va a circular por la red. Para finalizar, se muestra la duración del paquete en milisegundos.

```
Node          Time    Event
-----------------------------------------------------------------------------
```

Durante la simulación, Avrora devuelve tres tipos diferentes de paquetes. A continuación, se describirán cada uno de los mismos:

- **Paquetes Beacon**: Son paquetes de sincronización de nodos en la red, que almacenan los nodos vecinos del nodo transmisor. Por tanto, estos paquetes tienen longitud variable, por lo que cada vez tendrán una duración diferente. Como ejemplo, mostramos dos eventos de paquetes de beacon, un beacon emitido por el nodo 0 y un beacon recibido por el nodo 2. Marcados en color amarillo, destacamos los nodos vecinos que contiene este último beacon recibido por el nodo 2.

```
0  0:00:01.142141  ---->
00.00.00.0F.A7.13.41.88.00.22.00.FF.FF.00.00.18.00.00.40.00.00.00.00.E0.AC
0.786 ms
2  0:00:36.502066  <====
00.00.00.0F.A7.1F.41.88.28.22.00.FF.FF.00.00.18.04.23.00.00.00.00.00.00.03.F4.
00.02.FC.00.01.F4.00.04.F4.2D.E9  1.164 ms
```

- **Paquetes de datos**: Estos paquetes contienen los datos enviados por la aplicación, y que por tanto dependen de cada una de ellas. Sin embargo, con carácter general estos paquetes están formados por una cabecera, un conjunto de datos o *payload*, y un código CRC. La cabecera tiene un tamaño de 10 bytes (5 bytes de cabecera más un preámbulo de 5 bytes). El *payload* tienen un tamaño también constante de 29 bytes y el CRC de 2 bytes. En total, una trama de datos tiene un tamaño de 41 bytes, y una duración de 1.290ms. Un ejemplo de un paquete de datos transmitido por el nodo 4 y recibido por el nodo 1 se muestra a continuación:

```
   4  0:00:37.101902  ---->
00.00.00.0F.A7.23.61.88.10.22.00.00.00.04.00.17.00.00.00.00.00.04.08.EE.00.0
4.00.08.00.00.00.00.CA.FE.00.00.00.00.00.A7.FB  1.290 ms
   1  0:00:37.101934  <====
00.00.00.0F.A7.23.61.88.10.22.00.00.00.04.00.17.00.00.00.00.00.04.08.EE.00.0
4.00.08.00.00.00.00.CA.FE.00.00.00.00.00.2D.E6  1.290 ms
```

La cabecera y preámbulo está formado por los siguientes bytes, marcados en color verde en la trama anterior:

- **00.00.00.0F**: preámbulo fijo para todos los paquetes TinyOS.
- **A7**: Byte de sincronización.
- **23:** Tamaño de la cabecera sin preámbulo más los datos. Pasando a decimal este campo, tenemos un valor de 35 bytes, equivalente a 0x23 (en hexadecimal).
- **61**: Identificador de la placa de sensores.
- **88:** Identificador del tipo de nodo sensor.
- **10:** es el campo que representa al identificador del mensaje activo (AM_CTP_DATA=0x23), que se utiliza para permitir enviar paquetes de distintas aplicaciones.

- **22:** Identificador de grupo, para crear una red virtual.

Los datos del paquete están formados por los siguientes 29 bytes (marcados en color azul en la trama anterior), que son específicos de la aplicación.

- **00.00**: Dirección de destino del paquete. Las direcciones de TinyOS se especifican utilizando dos bytes, por lo que el número máximo de nodos direccionables sería 256.
- **00.04:** Dirección de origen del paquete.
- **00.17:** es el campo que representa la longitud de los datos de la aplicación del mensaje (23 en decimal), es decir, el número de bytes entre el siguiente a este campo hasta el final del mensaje (sin contar el CRC).
- **00.00.00.00.00.04.08.EE.00.04.00.08.00.00.00.00.CA.FE.00.00.00.00.00:** payload del paquete de datos.

Finalmente, se incluye un código de verificación (checksum) o CRC del paquete, compuesto de los dos últimos bytes (marcados en color morado). El CRC de los paquetes mostrados serían A7.FB y 2D.E6.

El tiempo de transmisión de estos paquetes es de 1.290 ms y es siempre constante (ya que depende de la longitud de la trama que se transmite por la red). El tiempo se calcula como: T=Longitud del mensaje/Bandwidth.

- **Paquetes ACK:** Estos paquetes se transmiten por el nodo receptor de un mensaje de datos y se dirigen al nodo emisor, para permitir la confirmación de que un paquete de datos ha llegado correctamente a su destino. Siempre tienen el mismo tamaño y por tanto su tiempo de transmisión es siempre el mismo (0.346 ms).

```
0  0:00:37.102708  ----> _[1;00m00.00.00.0F.A7.05.02.00.10.51.7C  0.346 ms
2  0:00:37.102740  <==== _[1;00m00.00.00.0F.A7.05.02.00.10.2D.DF  0.346 ms
1  0:00:37.102740  <==== _[1;00m00.00.00.0F.A7.05.02.00.10.2D.EA  0.346 ms
4  0:00:37.102740  <==== _[1;00m00.00.00.0F.A7.05.02.00.10.2D.EA  0.346 ms
```

Una vez se termina la simulación, el monitor *packet* muestra un resumen de todos los paquetes que se intercambian durante dicha simulación, incluyendo el número de paquetes (la suma global) transmitidos y recibidos en la red. Lo podemos ver tanto en bits como en paquetes. La información está desglosada por número de nodo, y para cada uno de ellos podemos ver los bits y paquetes enviados, los recibidos, los bits que estaban corruptos y los paquetes que se han perdido en el tránsito. El siguiente ejemplo muestra este resumen del monitor *packet* para una simulación de 5 nodos:

```
=={ Packet monitor results }=================================================
Node     sent (b/p)          recv (b/p)    corrupted (b)   lostinMiddle(p)
-----------------------------------------------------------------------------
   0      1922 / 78           3546 / 97            148       4
   1      1360 / 44           2875 / 97            119       3
   2       763 / 20           3178 / 110           118       4
   3       848 / 22           3069 / 111             0       0
   4       767 / 20           3240 / 115            30       2
```

4.2.2 Monitor *energy*

El monitor *energy* muestra, al final de la simulación, los resultados de consumo energético de todos los nodos, incluido el gateway. El consumo se muestra para cada uno de los distintos componentes del nodo sensor. En primer lugar aparecen datos relativos al nodo, y se muestra su tiempo de vida tanto en ciclos como en segundos (tiempo de simulación). Por ejemplo, el siguiente fragmento muestra los datos relativos al nodo 1.

```
=={ Energy consumption results for node 1 }=================================
Node lifetime: 294912000 cycles,  40.0 seconds
```

A continuación aparece el consumo energético de la CPU, en Julios, y para cada uno de los estados de energía en los que puede incurrir. Además, se muestra el tiempo en ciclos de reloj que el nodo ha gastado en ese estado. Por ejemplo, el siguiente fragmento muestra que el nodo 1 ha estado en los modos activo, idle y standby, así como el consumo y el tiempo en ese estado. Obviamente, la suma de los ciclos en los distintos estados de cada componente debe coincidir con el número de ciclos totales de una simulación.

```
CPU: 0.5300189933772786 Joule
   Active: 0.4494971410164388 Joule, 145992860 cycles
   Idle: 0.07094159370800782 Joule, 52147896 cycles
   ADC Noise Reduction: 0.0 Joule, 0 cycles
   Power Down: 0.0 Joule, 0 cycles
   Power Save: 0.0 Joule, 0 cycles
   RESERVED 1: 0.0 Joule, 0 cycles
   RESERVED 2: 0.0 Joule, 0 cycles
   Standby: 0.0 Joule, 0 cycles
   Extended Standby: 0.009580258652832031 Joule, 96771244 cycles
```

Análogamente, se ofrece la misma información para los LEDs en los estados ON y OFF.

```
Yellow: 0.23040358927408855 Joule
   off: 0.0 Joule, 37530245 cycles
   on: 0.23040358927408855 Joule, 257381755 cycles
Green: 0.23040678149414065 Joule
   off: 0.0 Joule, 37526679 cycles
   on: 0.23040678149414065 Joule, 257385321 cycles
Red: 0.23041938655598962 Joule
   off: 0.0 Joule, 37512598 cycles
   on: 0.23041938655598962 Joule, 257399402 cycles
```

En el caso de estos sensores, es muy importante conocer el consumo energético del dispositivo radio que permite la comunicación con el resto de nodos sensores que forman parte de la red. Como se ha comentado anteriormente, la radio es el dispositivo que habitualmente requiere mayores recursos de energía. A continuación se muestra el consumo para la radio, en los estados Power Off, Power Down, Idle, Recepción y Transmisión.

```
Radio: 1.0849852463263268 Joule
   Power Off:            : 9.341787027994792E-7 Joule, 114791879 cycles
   Power Down:           : 2.868330729166667E-4 Joule, 35246048 cycles
   Idle:               : 5.411841528320313E-4 Joule, 3122099 cycles
   Receive (Rx):         : 1.0815538564453127 Joule, 141384402 cycles
   Transmit (Tx):        31:   : 0.0026024384765625 Joule, 367572 cycles
```

La placa sensora también tiene un apartado específico dentro de la información de consumo de energía.

```
SensorBoard: 0.08399999999999999 Joule
   on:  : 0.08399999999999999 Joule, 294912000 cycles
```

Para finalizar, se incluye el desglose del consumo para la memoria flash externa que tiene el nodo sensor.

```
flash: 2.4E-4 Joule
   standby: 2.4E-4 Joule, 294912000 cycles
   read: 0.0 Joule, 0 cycles
   write: 0.0 Joule, 0 cycles
   load: 0.0 Joule, 0 cycles
```

Esta información se repite para todos los nodos sensores que forman parte de la red de sensores simulada.

5. Diseño

En este capítulo se describe el diseño de un módulo que se va a integrar con el simulador Avrora, de acuerdo al análisis explicado en el capítulo 3, cuyo objetivo va a ser el de proporcionar una interfaz gráfica que visualice los aspectos más importantes de una simulación de una red de sensores en tiempo de ejecución. El módulo que hemos implementado en este Proyecto Fin de Carrera se denomina `avInterfaz`. Este capítulo describe el diseño de clases realizado, su funcionalidad y el diseño de la interfaz.

5.1 Diseño de la Interfaz

Como ya se ha comentado, el fin de este proyecto es la implementación de un módulo que pueda integrarse con el simulador Avrora y que ejecute, de forma simultánea, una interfaz gráfica que permita visualizar de una manera sencilla e intuitiva los siguientes aspectos relevantes:

1. La topología de la red, es decir, la ubicación física en la que se encuentran desplegados los nodos sensores.
2. Las comunicaciones de la red, es decir, qué nodos hablan con qué otros nodos dentro de la red.
3. El análisis del tráfico transmitido en una simulación, más concretamente, el número de paquetes transmitidos por los nodos (número de tanto de datos, *beacon* y *acks* que han intercambiado los nodos) en una simulación completa.
4. El análisis del consumo de cada nodo de red en una simulación, y más concretamente, por cada componente físico y cada estado. Dado que el componente de mayor consumo en un nodo sensor es el componente radio[ref], sólo se tendrá en cuenta el consumo debido a las comunicaciones de la radio.

Ya que Avrora muestra los resultados de la simulación del comportamiento de una red de sensores inalámbricos en modo texto, se ha pretendido presentar la información de forma gráfica sencilla y de fácil comprensión.

La interfaz gráfica presenta varias pestañas que se podrán seleccionar mientras se está llevando a cabo la simulación y que muestran en tiempo de simulación la información de manera gráfica. La vista principal del módulo implementado muestra las siguientes pestañas:

- Datos simulación.
- Tráfico y Energía.
- Despliegue inicial.
- Despliegue final.

La pantalla principal se muestra en la siguiente figura, donde aparece en la parte superior las pestañas mencionadas. A continuación se describe la funcionalidad y el diseño cada una de ellas.

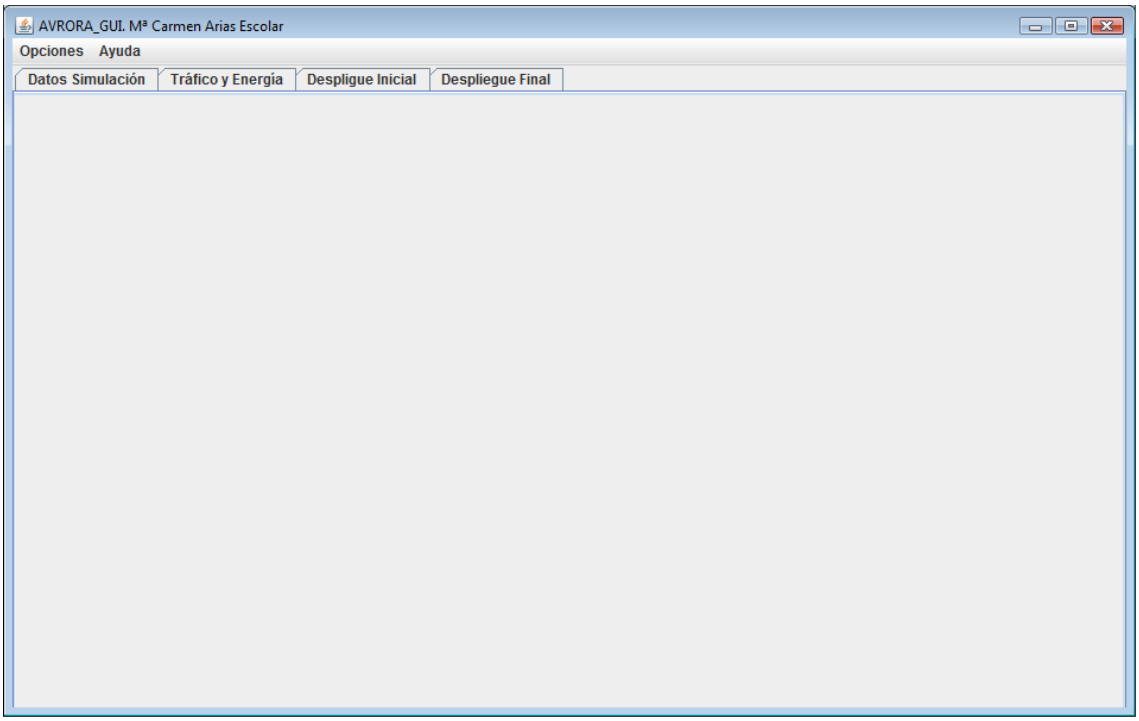


Ilustración 40 Pantalla principal de la ejecución del módulo impelemtado

5.1.1 Datos de la simulación

La simulación de una red de sensores precisa que se indiquen una gran cantidad de parámetros. Los parámetros más importantes se han resumido y mostrado al usuario en una de las pestañas de la interfaz gráfica. Esta pestaña recibe el nombre de "Datos de la simulación" y presenta la siguiente información:

- Número de nodos: Total de nodos que intervienen en la simulación.
- Radio de cobertura de los nodos: Alcance de la radio de los nodos, es decir, distancia en metros máxima que la radio puede alcanzar.
- Tiempo de simulación: Tiempo, especificado en segundos, durante el que se estará monitorizando el comportamiento de la red de sensores.
- Plataforma de nodos sensores: Indica el modelo de hardware de nodo que se está usando en la simulación. Aunque se soportan varios modelos, habitualmente, se utilizará micaz.
- Monitores que se simulan: Avrora es capaz de integrar varios monitores en una simulación de manera que la ejecución del simulador pueda personalizarse en función de las necesidades. Nuestro módulo, usará necesariamente los monitores "energy" (energía) y "packet" (paquetes).
- Tipo de topología: Indica la topología de la red de sensores. Avrora permite dos tipos de topología de entrada: 1) simular una red donde las ubicaciones de los nodos puedan especificarse, para lo cual se usará la opción `-topology=static` y se proporcionará el nombre de un fichero en la opción `–topology-file` o 2) no se especificará la opción `–topology` en cuyo caso el despliegue de los nodos será aleatorio.

- Fichero de topología: es el fichero que muestra la ubicación de los nodos en un espacio de tres dimensiones. La ubicación viene indicada por las coordenadas *x,y,z*. Un ejemplo de fichero de topología es el siguiente:

```
node0 0 0 0
node1 15 0 0
node2 5 5 0
node3 10 10 0
node4 15 10 0
```

  donde el `node0` representa el nodo gateway.

- Ruta de almacenamiento del fichero de *logs*: Todos los resultados de la simulación se almacenan en un fichero *.txt dentro de una ruta de red. Esta ruta se especifica en la pantalla de "Datos de la simulación", de modo que el usuario final sepa la ubicación del fichero en caso de querer realizar algún análisis específico.

  Esta es la vista que tenemos cuando seleccionamos la pestaña de *Datos Simulación:*

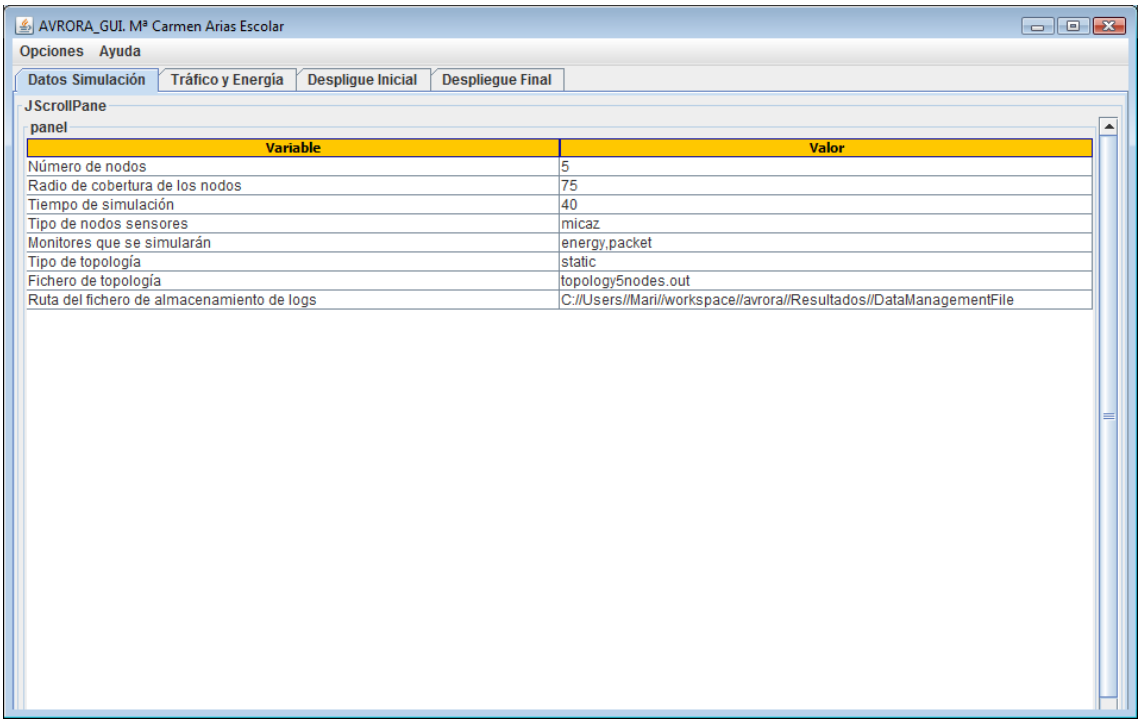


Ilustración 411 Pantalla "Datos Simulación"

5.1.2 Tráfico y consumo energético

Una vez finalizada la simulación de la red de sensores, se pueden mostrar los datos, por nodo, relativos al tráfico cursado y el consumo energético asociado. Inicialmente, esta información se mostraba en dos pestañas separadas. Ya que el objetivo de este proyecto es facilitar la interpretación de los datos asociados a la simulación, finalmente, se decidió integrar ambas vistas en una sola, de modo que con una única tabla, se podría ver el número de paquetes emitidos/recibidos y de qué tipo eran por cada nodo, y el consumo energético de los mismos, ya que el consumo energético está relacionado con la actividad que el nodo tenga en la red, lo más lógico era mostrar ambos casos en una misma tabla.

Dado que la simulación se puede realizar con gran cantidad de nodos, se ha incluido un desplegable en el que se seleccionará el nodo del que se quiere mostrar la información de tráfico y de consumo energético.

Inicialmente, cuando se selecciona esta vista y antes de proceder a la simulación, aparece seleccionado por defecto el nodo gateway (nodo 0), y todos los contadores a cero, como se puede comprobar en la siguiente figura:

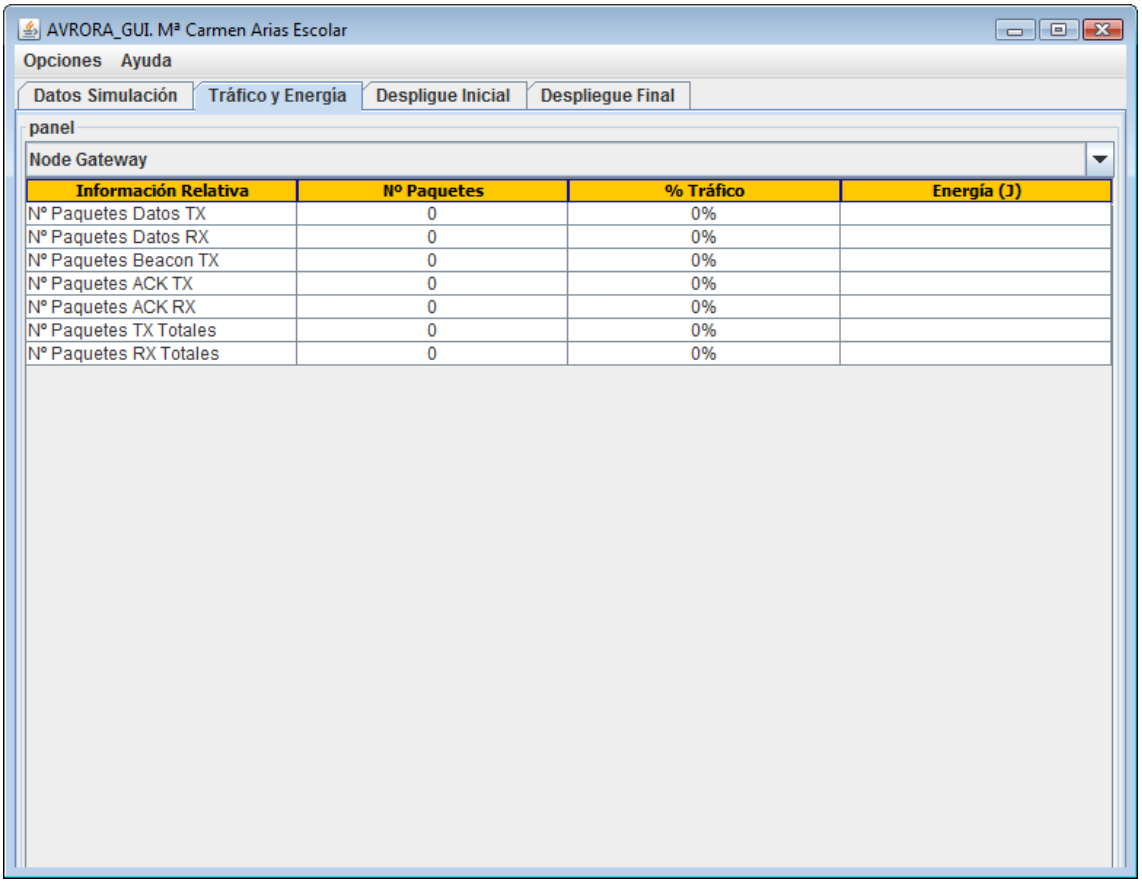


Ilustración 42 Pantalla "Tráfico y energía"

En la parte superior de la tabla en la que se muestra la información de tráfico y de energía consumida se ha incluido un menú desplegable en el que se puede seleccionar el nodo del cual queremos que se muestren los datos, tal y como se puede ver en la siguiente figura, en la que la red a simular estaba compuesta por cuatro nodos sensores más el nodo gateway.

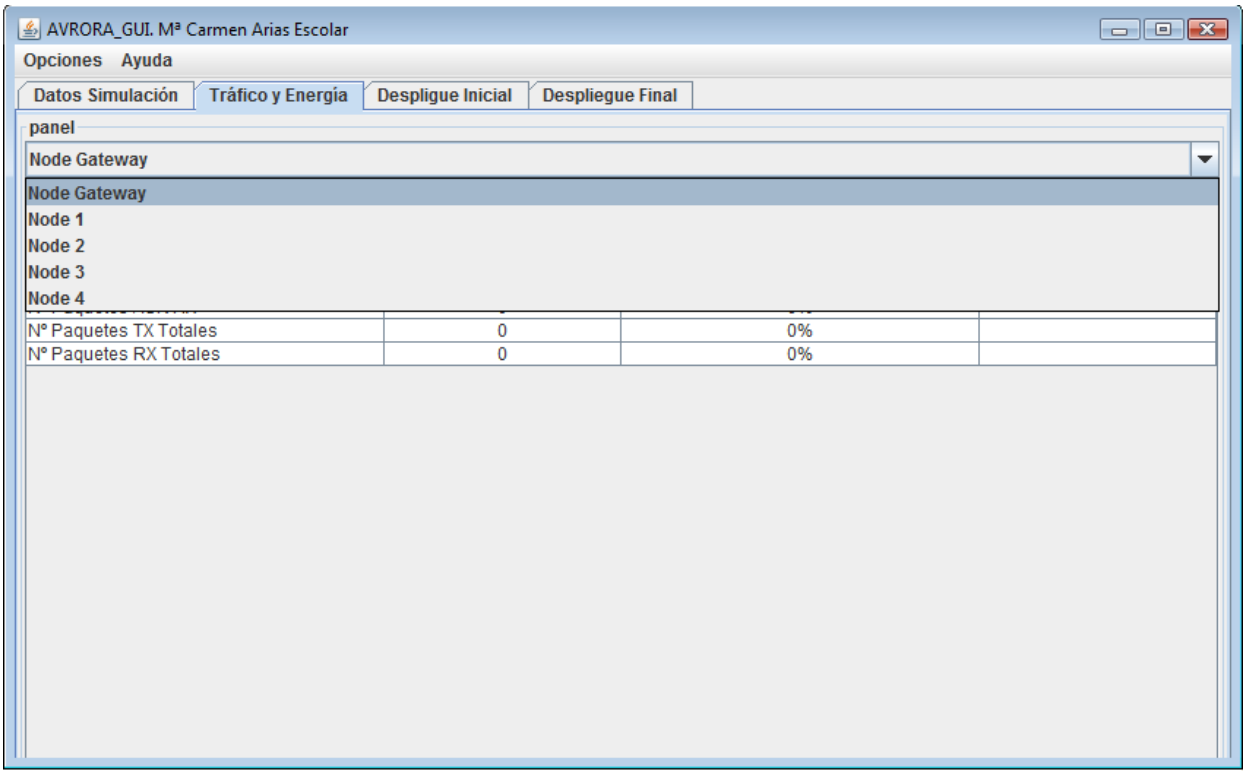


Ilustración 43 Pantalla "Tráfico y energía" desplegable de los nodos

Según avanza la ejecución, y seleccionamos el nodo sensor que se quiere analizar, los valores de paquetes intercambiados con los otros nodos, el porcentaje de tráfico que éstos representan con respecto al total de la red se actualizan, tal y como se puede ver en las siguientes figuras. El valor de la energía total consumida por la radio tanto en modo transmisión como en recepción no se actualiza hasta que ha finalizado completamente la simulación, dado que el simulador Avrora no proporciona datos parciales de la energía consumida en los componentes.

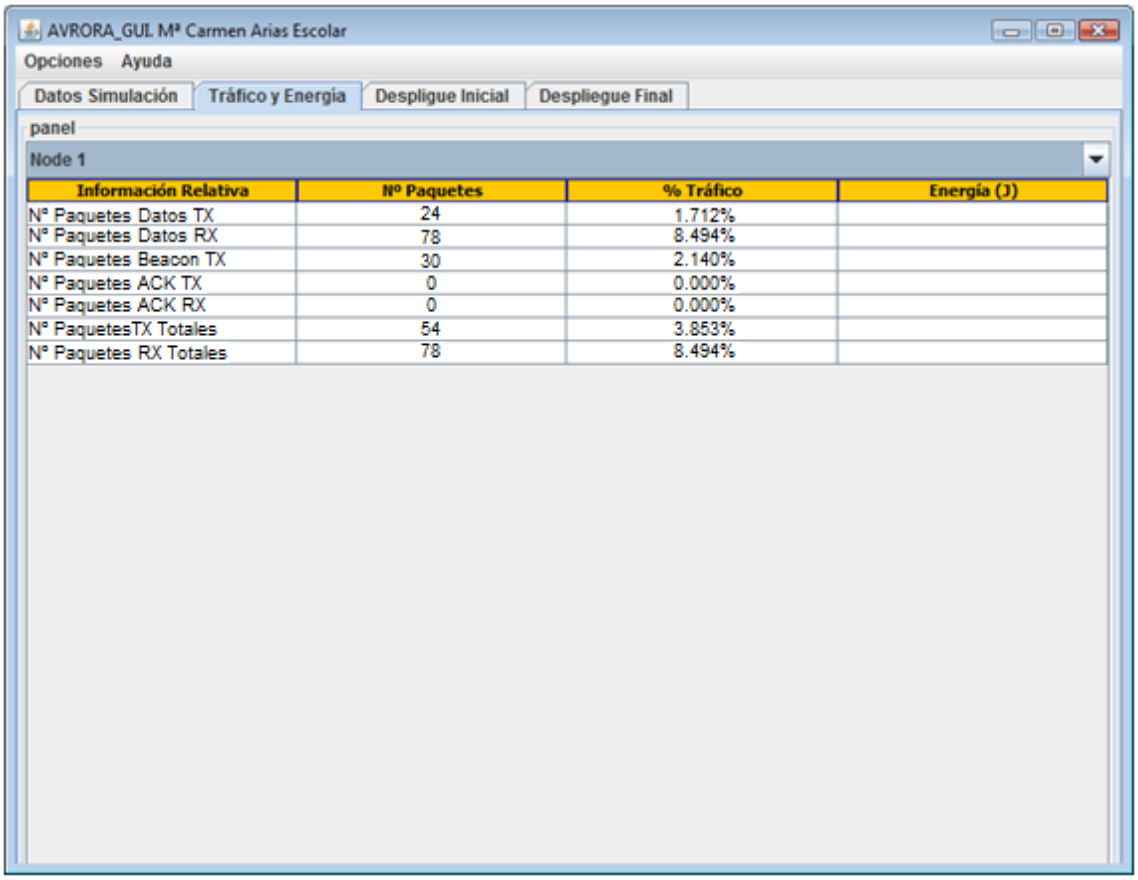


Ilustración 44 Pantalla "Tráfico y energía" - Valores parciales nodo sensor 1

Como se puede ver en la Ilustración 44, esta pantalla muestra además el porcentaje del tráfico debido a cada uno de los tipos de paquetes considerados en esta simulación con respecto al número total de paquetes transmitidos y recibidos por todos los nodos de la red. Esta información es útil para identificar los nodos más activos en la red, y que posiblemente puedan ser puntos críticos potenciales (cuellos de botella, agotamientos de las baterías, etc.).

Al término de la simulación esta pantalla se completará con la información relacionada a la energía en Julios consumida por cada nodo, debido a la transmisión y recepción de paquetes. La Ilustración 45 muestra dicha información en la última columna de la tabla para el nodo 1 de la simulación.

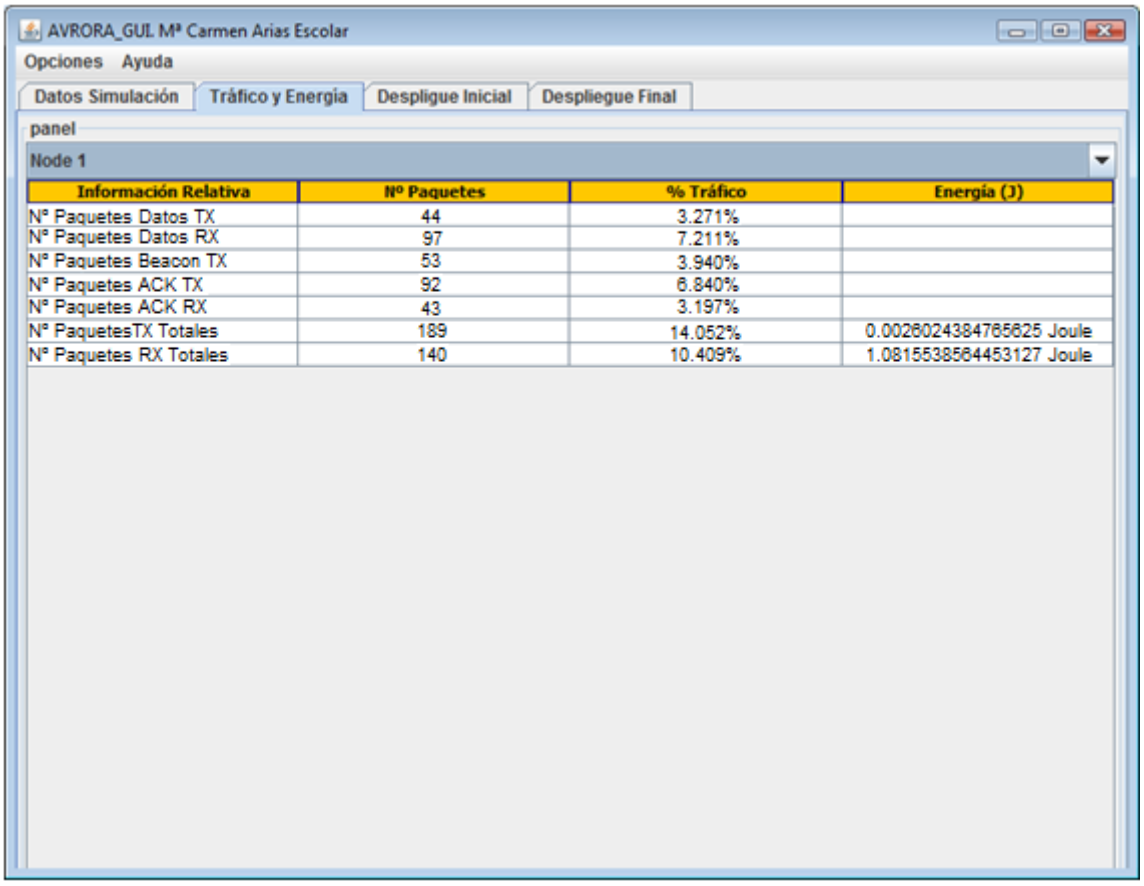


Ilustración 45 Pantalla "tráfico y energía" resultados finales nodo sensor 1

5.1.3 Despliegue Inicial

Dado que, como parámetro de ejecución de la simulación, aportamos un fichero en el que se indica la cantidad de nodos que se están teniendo en cuenta para la misma y en el que también se incluye cómo están distribuidos en el espacio, se puede implementar una vista en la que representar, en un plano de dos dimensiones, la localización de los mismos.

La representación de los nodos se ha realizado en dos colores, para diferenciar los distintos roles que los nodos pueden tener. El color rojo, con el que se representa el nodo gateway, y el azul, en el que se representan el resto de nodos sensores en la red.

Junto a todos los nodos, independientemente del tipo de nodo del que se está hablando, se han incluido las coordenadas cartesianas en tres dimensiones, dado que así se especifica en el fichero de topología. En la siguiente figura se muestra un ejemplo de despliegue inicial para una simulación de 4 nodos sensores más el nodo gateway. En este caso, se puede comprobar que el nodo gateway, representado en color rojo, se encuentra en la posición (0, 0, 0), es decir, en el origen del plano de coordenadas. Los cuatro nodos sensores, representados en color azul, se encuentran en las posiciones (5, 5, 0), (15, 0, 0), (10, 10, 0) y (15, 10, 0).

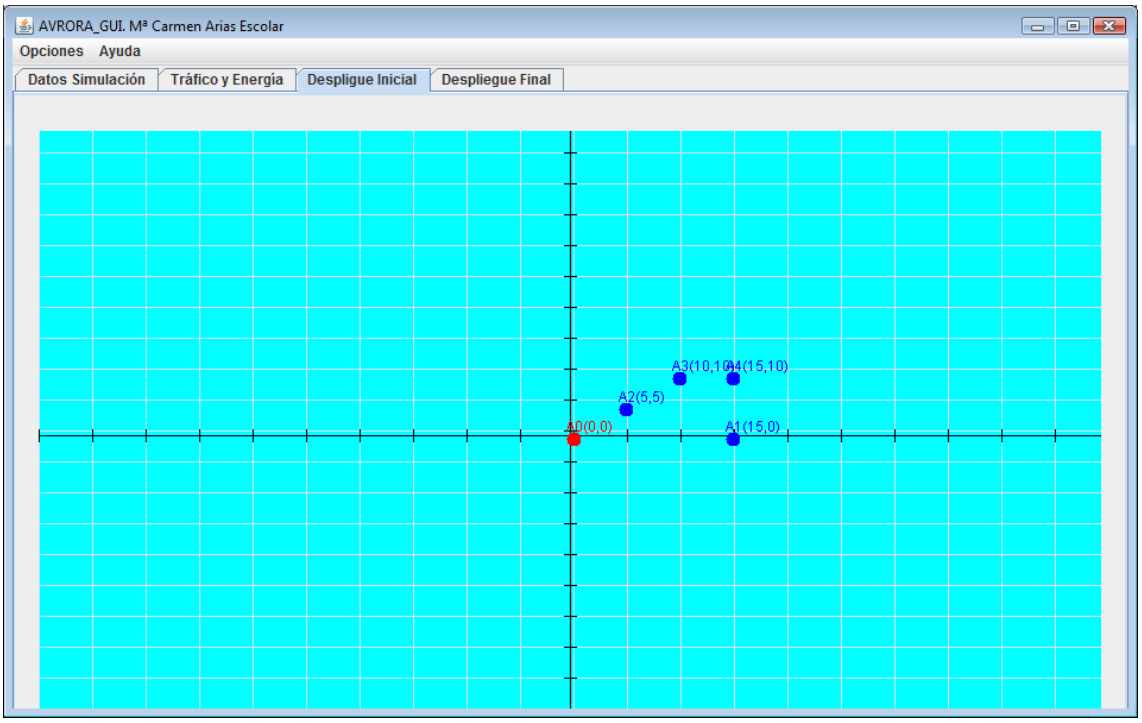


Ilustración 46 Pantalla "Despliegue inicial"

Sin embargo, la topología inicial se puede ver modificada durante la simulación. La topología final resultante dependerá de la operación del protocolo de enrutamiento ejecutado en los nodos sensores, que determinará el siguiente salto para encaminar un paquete de datos. La implementación de las aplicaciones de los nodos sensors queda fuera del alcance de este proyecto; desde el punto de vista de nuestro módulo la aplicación a ser ejecutada en los nodos es un parámetro de entrada más de nuestra simulación.

5.1.4 Despliegue final

En esta pestaña se representan las comunicaciones entre los nodos que forman parte de la red de sensores inalámbrica que se simula con Avrora, además de la topología final resultante.

Durante la simulación cada nodo emite paquetes *beacon* o de control con información del vecindario de cada nodo (aquellos que están dentro del alcance de su radio). Por tanto, es posible extraer información para establecer las vecindades de los nodos y el nodo siguente salto. En esta simulación, para 4 nodos sensores más un nodo gateway, todos los nodos eran capaces de hablar con todos los nodos sensores de la red, no perdiéndose la comunicación entre ellos durante el tiempo que ha durado la misma.

La siguiente pantalla muestra gráficamente la topología final resultante al término de la simulación. Asimismo, las líneas entre nodos muestran las decisiones de encaminamiento en cuanto al siguiente salto de cada nodo.

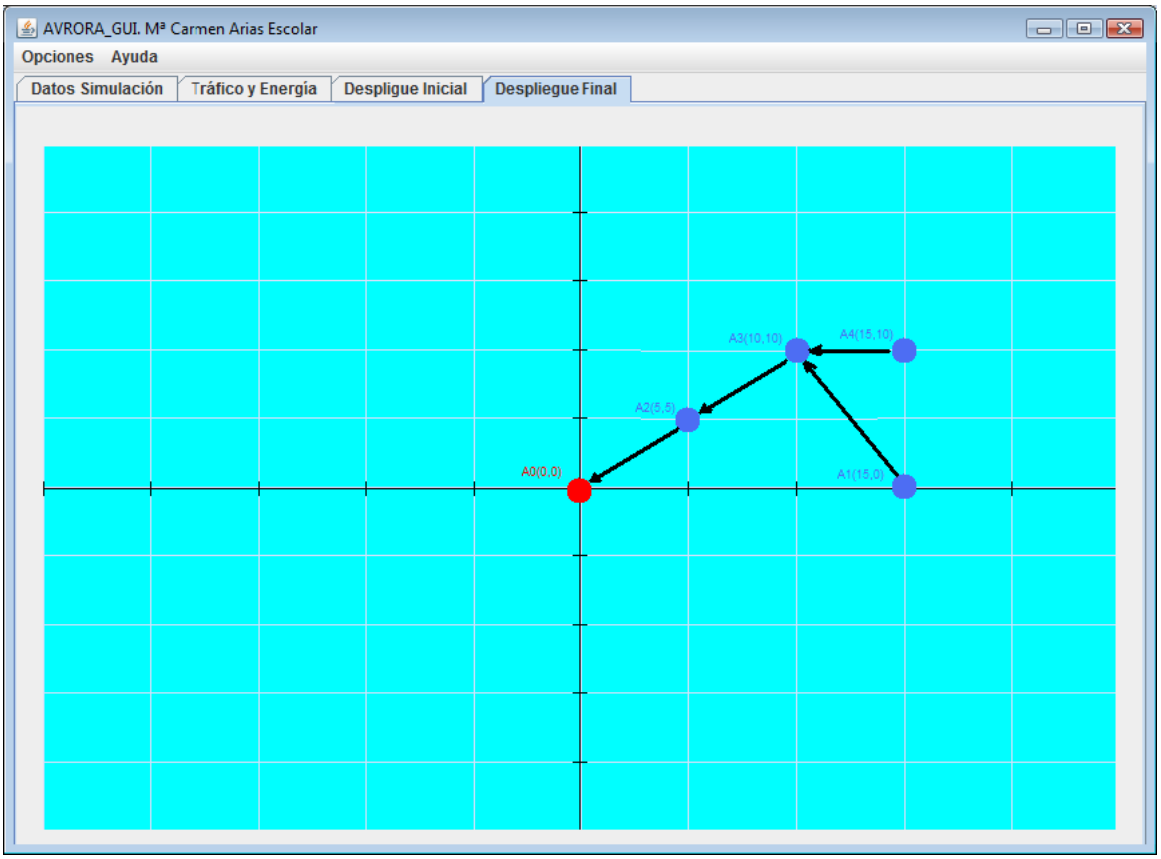


Ilustración 47 Pantalla "despliegue final"

5.2 Diseño de clases

En este apartado se define la solución llevada a cabo para el diseño de la aplicación con la que se representa de forma gráfica los resultados de la simulación de una red de sensores inalámbrica.

Como se puede observar en la siguiente figura, se ha incluido un nuevo paquete dentro del código original de Avrora, llamado ***avInterfaz***. El proyecto se ha desarrollado creando, dentro de este paquete, las siguientes cuatro clases:

- **Avrora_GUI.java:** representación gráfica de los datos de la simulación.
- **DataManagement.java**: procesado de los datos obtenidos durante la simulación.
- **PacketType.java**: definición de la tipología de los paquetes enviados y recibidos durante la simulación.
- **Plano2.java**: configuración de los planos de representación de la ubicación de los nodos sensores.

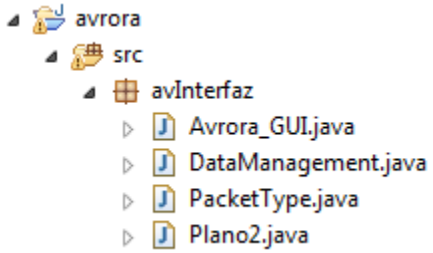


Ilustración 48 Estructura paquete *AvInterfaz*

Antes de proceder a explicar cada una de las clases, se estima oportuno explicar las modificaciones que se han incluido dentro del *Main.java* de Avrora, y que permiten la integración con las clases implementadas.

5.2.1 Main.java

Para poder tratar con todos los datos que se obtienen durante la simulación, en primer lugar, se crea un fichero en el que se va almacenando todo el tráfico que se está generando en nuestra red simulada. Ya que es necesario que este fichero se lea simultáneamente mientras se está almacenando información en él, se crea dentro de la clase *main* un hilo o thread que permita la ejecución de dos tareas simultáneamente. También aquí se solicita el cierre del hilo, una vez se ha terminado la simulación.

Los parámetros de entrada se analizan en esta clase. Por esto, es aquí donde se verifica si se ha incluido la condición execGUI de forma positiva (yes) o negativa (no). Si el parámetro está establecido a "yes", desde aquí se llama a `Avrora_GUI.java` y se lanza la interfaz gráfica. A continuación, se definirán las clases que forman parte del paquete **AvInterfaz**, indicando los atributos y métodos más importantes que se han implementado.

5.2.2 Avrora_GUI.java

En esta clase, se diseña la interfaz gráfica en la que se representan los datos de la simulación de la red de sensores. Con el método `AvroraGui` se implementa un frame o ventana que aparecerá como un pop-up al iniciar la simulación de Avrora. Dado que la información a mostrar precisa que se tengan diferentes vistas, se decide incluir dentro de la ventana una serie de pestañas, una por cada una de las representaciones de las que se quiere tener visión. Para la inclusión de pestañas dentro del frame, se usa el método `JTabbedPane`, y de define un `JPanel` para cada una (panel de datos de simulación, Tráfico y consumo energético, Despliegue Inicial y Despliegue final).

Cada una de las vistas que se tienen en la interfaz gráfica se puede seleccionar gracias a los botones de selección incluidos para cada una. Estos botones se han creado usando el método `JButton` que se encuentra dentro del paquete **javax.swing*.** La selección de los botones se realiza con un click de ratón. Para que esto funcione, es necesario activar los eventos de ratón, y así poder cambiar de pestaña de modo manual, por lo que se incluye el uso del método `actionListener` sobre los botones que se definen para cada pestaña.

La primera de las pestañas que se puede mostrar es la correspondiente a los datos de simulación usados para la ejecución de Avrora. Para ello, se crea un método dentro de esta clase llamado `panelDatosSimulacion()` en la que se crea una tabla alimentada por los parámetros de entrada. Los parámetros de entrada se almacenan en la matriz de argumentos que se define en el método `main` de Avrora.

Para la segunda pestaña, en la que se muestra para cada nodo el número de paquetes enviados/recibidos y la energía consumida por el nodo durante la simulación. Para que la vista sea clara, se crea una tabla en la que se mostrarán el número total de mensajes transmitidos y recibidos de datos, de beacon, y ACKs, así como el porcentaje de tráfico que representan, y la energía consumida representada en julios El número de paquetes transmitidos y recibidos dependiendo del tipo de mensaje del que se trata, se calcula en la clase `DataManagement.java`, que describiremos en el apartado correspondiente.

Con respecto a la energía, este valor se calcula en el monitor `energy` que se reporta durante la ejecución. Una vez se ha superado el tiempo de ejecución, desde la clase `Avrora_GUI.java` se lanza una petición a la clase `PacketMonitor.java`, dónde se obtienen los valores de energía consumida por la radio durante la emisión de los paquetes, y la gastada por la radio durante la recepción de los mismos.

En la tercera pestaña se puede ver la topología inicial o despliegue de la red de sensores inalámbricos que se va a simular. Desde aquí se hace una llamada a la clase `Plano2.java`, que es la encargada de dibujar los planos cartesianos, así como los nodos sensores que forman parte de la simulación. Este proceso es el mismo que se lleva a cabo para representar los puntos en la cuarta pestaña.

5.2.3 DataManagement.java

Como su nombre indica, esta clase se encarga del tratamiento de los datos que se obtienen durante la simulación de la red de sensores inalámbricos. Esta información es tomada directamente de la salida en modo texto proporcionada por el simulador Avrora.

El objetivo de este proyecto es conocer cuánta información, y de qué tipo, se ha transmitido por la red simulada. Con el fin de tener bien ordenada la información, se crean al principio de esta clase las siguientes variables, en las que almacenaremos los datos obtenidos del análisis de los paquetes que emiten o reciben los nodos que forman la red de sensores. Conviene recordar que los nodos pueden enviar y recibir paquetes de tipo “dato”, tipo “beacon” y tipo “ACK”.

- `dataMatrixEnvio[][]`: En esta matriz, de tamaño n x n, donde n es el número de nodos sensores por los que está formada la red, se almacena el número de paquetes de datos que se envían entre los nodos. El índice de fila indica el nodo origen del paquete, y el índice columna indica el nodo destino del paquete.
- `dataMatrixRecibo[][]`: En esta matriz, de tamaño n x n, donde n es el número de nodos sensores por los que está formada la red, se almacena el número de paquetes de tipo “datos” que reciben los nodos. El índice de fila indica el nodo que recibe el paquete, y el índice columna indica el nodo que ha enviado el paquete. Remarcar que es posible que las matrices `dataMatrixEnvio` y `dataMatrixRecibo` no contengan la misma información, hecho que puede suceder si hay pérdida de paquetes en la red (paquetes que han expirado su tiempo de vida TTL, corruptos, etc).
- `beaconMatrixEnvio[][]`: En esta matriz, de tamaño n x n, donde n es el número de nodos sensores por los que está formada la red, se almacena el número de paquetes beacon que se envían entre los nodos. El índice de fila indica el nodo origen del paquete, y el índice columna indica el nodo destino del paquete.
- `beaconMatrixRecibo[][]`: En esta matriz, de tamaño n x n, donde n es el número de nodos sensores por los que está formada la red, se almacena el número de paquetes beacon que reciben los nodos. El índice de fila indica el nodo que recibe el paquete, y el índice columna indica el nodo que ha enviado el paquete. Remarcar que es posible que las matrices `beaconMatrixEnvio` y `beaconMatrixRecibo` no contengan la misma información, hecho que puede suceder si hay pérdida de paquetes en la red (paquetes que han expirado su tiempo de vida TTL, corruptos, etc).

- `ackMatrixEnvio[][]`: En esta matriz, de tamaño n x n, donde n es el número de nodos sensores por los que está formada la red, se almacena el número de paquetes de confirmación de datos, o ACK, que se envían entre los nodos. El índice de fila indica el nodo origen del paquete, y el índice columna indica el nodo destino del paquete.
- `ackMatrixRecibo[][]`: En esta matriz, de tamaño n x n, donde n es el número de nodos sensores por los que está formada la red, se almacena el número de paquetes de confirmación de datos, o ACK, que reciben los nodos. El índice de fila indica el nodo que recibe el paquete, y el índice columna indica el nodo que ha enviado el paquete. Remarcar que es posible que las matrices `ackMatrixEnvio` y `ackMatrixRecibo` no contengan la misma información, hecho que puede suceder si hay pérdida de paquetes en la red (paquetes que han expirado su tiempo de vida TTL, corruptos, etc).

A partir del tratamiento de la información que contienen los paquetes beacon, podemos extraer las relaciones que existen entre los nodos. Por esto, se crea otra matriz, a la que se ha llamado `MatrizVecinos`, en la que se almacenan, para cada nodo, los nodos que son visibles y accesibles. Los paquetes beacon tienen una longitud variable porque, según va avanzando la simulación, la información acerca de los nodos con los que mantiene comunicación se incluye en los mismos.

En la clase principal `main` de Avrora se ha creado un fichero en el que se almacenan los paquetes de información que se transmiten por la red durante la simulación de nuestra red de sensores inalámbricos. En `DataManagement.java` se crea una instancia `fileReader` de modo que se accede a este fichero y se va leyendo una a una las líneas introducidas. Cada una de estas líneas se procesa, y se extrae la información necesaria para saber si el paquete enviado es de datos, de beacon, o de ack. También se extrae la información sobre qué nodo ha enviado el paquete, y hacia qué nodo va dirigido, y con estos datos, se rellenan las matrices de envío y recibo para cada tipo de paquete. Adicionalmente, si estamos ante un paquete beacon, se actualizan los datos de la matriz de vecinos.

5.2.4 PacketType.java

Se trata de una clase meramente descriptiva, que básicamente se utiliza para la inicialización y definición de las variables necesarios para el tratamiento de paquetes. En ella, se han definido los tipos que tienen las variables usadas para el tratamiento de los paquetes que se intercambian durante la simulación de una red de sensores (`numACK`, `numData` y `numBeacon`). Una vez definidos los tipos de las variables, se inicializan para que puedan ser utilizadas por los métodos de otras clases.

5.2.5 Plano2.java

En esta clase se dibujan los planos cartesianos representados en las vistas “Despliegue inicial” y “Despliegue final”. Para poder representar un plano cartesiano, se usa el método `paint`, con el que se puede establecer el color de fondo del plano (en nuestro caso, en cyan), y pintar tanto los ejes y las separaciones (de 5 unidades en 5 unidades). La ubicación de los nodos sensores se le da a Avrora en el fichero de topología que se indica en los atributos a la hora de iniciar la simulación. El formato que sigue el fichero de topología es:

```
identificador_nodo coordenada_x coordenada_y coordenada_z
```

Se decide usar el método `FileReader` para poder acceder al fichero y el método `readLine` para tratar el mismo línea por línea, o lo que es lo mismo, nodo por nodo.
Una vez tenemos almacenada la línea con la información de la ubicación espacial del nodo sensor, separamos la información de la línea y representamos los valores "x" (ubicación horizontal) e "y" (ubicación vertical).

Con un objeto de la clase `Graphics`, usada para la representación del plano, se pueden seleccionar los colores con los que representar los nodos. Por esto, se decide usar el método `setColor` sobre este objeto con "RED" (rojo) para el nodo gateway, y el "BLUE" (azul) para el resto de nodos.

5.3 Integración con Avrora

Tal y como se ha comentado con anterioridad, la inserción de una interfaz gráfica permite disponer de una forma más amigable los resultados de las simulaciones. Con el fin de impactar lo menos posible en el código original de Avrora, la ejecución o no de la interfaz gráfica se define en los parámetros de entrada de la simulación con la inclusión de un nuevo parámetro denominado `-execGUI` y que puede tomar los valores "yes" (se lanza la interfaz gráfica) o "no" (no se lanza la interfaz gráfica). El siguiente ejemplo de ejecución muestra cómo incorporar el flag en el comando de ejecución de Avrora.

```
-action=simulate -simulation=sensor-network -nodecount=1,4 -
platform=micaz -input=elf -seconds=100 -report-seconds -
monitors=energy,packet-radio-range=75 -topology=static -
execGUI=yes -topology-file=topology5nodes.out
TestNetworkLPLBS.elf TestNetworkLPLNodes_50dc_20%.elf
```

Si el argumento de entrada está definido como "yes", desde `Main.java` se llama a un método denominado `DataProcessingThread` que se ha incluido dentro de la clasa principal para el control del hijo de procesamiento de datos. Este nuevo hilo permite la lectura de ficheros mientras éste está siendo usado para la escritura, y además, se solicita la ejecución de la clase `Avrora_GUI` que se encuentra dentro de nuestro paquete **AvInterfaz**.

Si el flag se encuentra definido como "no", se ejecuta el método `runAction` definido en Avrora para la simulación en modo texto de la red de sensores inalámbricos, en su forma habitual.

6. Implementación

En este apartado se define la solución llevada a cabo para el diseño de la aplicación con la que se representa de forma gráfica los resultados de la simulación de una red de sensores inalámbrica.
Se ha incluido un nuevo paquete dentro del código original de Avrora, llamado **avInterfaz**. Como se comentó en el capítulo anterior, el paquete **avInterfaz** se integra de las clases mostradas en la Ilustración 48: DataManagement.java, Plano2.java, PacketType.java y Avrora_GUI.java.

A continuación, se definirá con detalle los métodos más importantes de los que constan estas clases.

6.1 Avrora_GUI.java

En esta clase, se ha implementado el código necesario para mostrar una interfaz gráfica amigable, tal y como se ha descrito en apartados anteriores. Esta clase extiende de `JFrame`, ya que es necesario que herede los métodos para la creación de ventanas y así poder diseñar nuestra interfaz gráfica, y también implementa los métodos de ChangeListener, precisos para activar los eventos de ratón y que el usuario final pueda interactuar con la interfaz gráfica. La definición de esta clase es la siguiente:

```
public class Avrora_GUI extends JFrame implements ChangeListener
```

Para el desarrollo de esta clase, se ha precisado de la inclusión de los siguientes métodos:

- `Avrora_GUI()`
- `PanelInicial()`
- `Botones()`
- `panelDatosSimulacion()`
- `tablaDatos()`
- `panelResumenTraficoYConsumoEnergetico(JPanel panelActual)`
- `completarTabla(int index, JTable tabla)`
- `panelDespliegueInicial()`
- `tablaDatos()`
- `seleccionBotones()`

6.1.1 Método Avrora_GUI

El método `Avrora_GUI` es un método público que no recibe parámetros y que implementa la interfaz, es decir, se ha programado la aparición de una nuevo marco en el que se muestran varias pestañas que visualizarán los datos de la ejecución. Primero, es necesario crear un *frame*, al que daremos un nombre que se mostrará durante toda la ejecución:

```
frame = new JFrame("AVRORA_GUI. Mª Carmen Arias Escolar");
```

Una vez creado el frame, se incluyen todas las pestañas de las que constará la interfaz gráfica. Primero se creará el objeto `TabbedPane` con el que se definen las mismas.

```
tabbedPane = new JTabbedPane();
```

Definido este objeto, se incluyen todas las pestañas de las que consta la interfaz, siguiendo este formato:

```
tabbedPane.addTab("Datos Simulación", panelDatos);
tabbedPane.setMnemonicAt(0, KeyEvent.VK_1);
```

6.1.2 Método JPanel PanelInicial

Este método público que no recibe parámetros de entrada y se usa para inicializar la ventana.

```
panelActual = new JPanel();
```

6.1.3 Método JPanel Botones

Con este método, se establece un botón en cada una de las pestañas que se han creado con el método anterior. Cuando se pulse un determinado botón, se abrirá la pestaña con la información que se quiere visualizar.

```
b1 = new JButton("Datos de la simulación");
```

Para activar una u otra pestaña, se usa el método addActionListener, como se muestra a continuación:

```
seleccionBotones eb = new seleccionBotones();
b1.addActionListener(eb);
```

6.1.4 Método panelDatosSimulacion

Con este método se muestran los datos de la pestaña “Datos Simulación”. Desde aquí se invoca al método `tablaDatos()` en la que se conforma y se rellena la tabla que se muestra en esta vista, de la siguiente manera:

```
panelActual = tablaDatos();
```

El método `tablaDatos()` usado para completar esta pestaña se define a continuación:

6.1.5 Método tablaDatos

Este es el método que se utiliza para completar la información de la ventana de datos de la simulación. En primer lugar, se crea una matriz de datos compuesta por dos columnas, una en la que se define el nombre del parámetro de entrada, y otra en la que se muestra el valor introducido para la simulación

```
Object[][] datos = new Object[][] {{"Número de nodos",
                        String.valueOf(avrora.Main.nodesNumber)},
          …
}
```

Los valores que se pasan para poder llevar a cabo la simulación se almacenan de forma pública en la clase principal de Avrora. Desde esta clase se puede acceder directamente a estos valores. Además de la matriz de datos se crea un vector que incluye los nombres de las columnas.

```
String[] cabeceras = new String[] {"Variable", "Valor" };
```

Una vez se tienen tanto la matriz de datos como el vector con los nombres de las columnas, se crea la tabla que se presentará en la vista de la pestaña de datos de la simulación:

```
JTable jtTabla1 = new JTable(datos,cabeceras);
jtTabla1.setMinimumSize(new Dimension(700, 700));
```

6.1.6 Método panelResumenTraficoYConsumoEnergetico

Este método recibe como parámetro el panel actual y a partir de la información contenida aquí se dibuja la tabla de tráfico y de consumo energético de cada uno de los nodos. Sólo se muestra la información de aquel nodo que se selecciona en un menú despegable. En este desplegable, el nodo 0 se denomina "nodo gateway". El combo tendrá *n* nodos, dónde *n* es el parámetro "nodeNames" que obtenemos a partir del número de nodos pasado como argumento al inicio de la simulación.

```
final JComboBox jcb = new JComboBox(nodeNames)
```

Creado el listado de nodos (`JComboBox` con el número de los nodos), indicamos que se complete la tabla con los datos del nodo que se haya elegido del listado. La tabla se completa llamando al método `completarTabla`(), descrito más adelante:

```
tabbedPane.addMouseListener(new MouseAdapter(){
      public void mouseClicked(MouseEvent me) {
      completarTabla(tabbedPane.getSelectedIndex(), tabla);
}
jcb.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e) {
      completarTabla(jcb.getSelectedIndex(), tabla);
}}
```

Se crea una tabla genérica que se irá rellenando con la información relativa al nodo que se quiere estudiar, una vez se haya seleccionado el nodo en el menú desplegable.

Con la siguiente línea se define el nombre de las columnas de la tabla:

```
final String[] columnNames = {"Información Relativa", "N°
Paquetes", "% Tráfico", "Energía (J)"};
```

Y la tabla de información tiene el siguiente formato, incluyendo una línea para el número de paquetes de datos transmitidos, número de paquetes de datos recibidos, número de

paquetes beacon transmitidos, número de paquetes ack transmitidos, número de paquetes ack recibidos, y otras dos líneas para las sumas de paquetes enviados y recibidos por el nodo.

```
Object[][] data = {{"N° Paquetes Datos TX", "", "", ""},
                   {"N° Paquetes Datos RX", "", "", ""},
                    …..
                  }
```

Para rellenar la tabla que se muestra en esta ventana, se invoca a método `completarTabla`, al que se le pasa como argumento el nodo del que queremos que se nos muestre la información y que hemos obtenido a partir del la lectura del nodo seleccionado en el combo de nodos que está incluido en esta ventana.

6.1.7 Método completarTabla

Este es el método en el que se completa la tabla que se muestra en la pestaña de resumen de tráfico y consumo energético por nodo. Para rellenar la información, este método precisa que se le pase por argumento el nodo del que se quiere incluir la información (`index`), así como la tabla que hay que rellenar (`tabla`).

En primer lugar, se obtiene la información de los paquetes enviados y recibidos de cada tipo, y se incluyen en la celda correspondiente:

```
for(int i=0; i<avrora.Main.nodesNumber; i++){
      pDatosTX+=DataManagement.dataMatrixEnvio[i][index];
      …..
}
```

En este método también se va llevando un conteo de los paquetes de cada tipo que ha enviado o recibido el nodo del que se está mostrando la información. Esto se debe a que, además de mostrarse el número parcial o total de paquetes enviados o recibidos por el nodo (dependiendo del momento de la ejecución en el que se ha pedido que se muestre la tabla) también se muestra el porcentaje que este valor representa con respecto al global de paquetes que han circulado por la red.

```
if(totalPaquetes!=0){
      float trafico0= pDatosTX/totalPaquetes;
      ……
      tabla.setValueAt(trafico0, 0, 2);
      ……
}
```

6.1.8 Método panelDespliegueInicial

Con este método se invoca a la clase `Plano2.java`, encargada de representar un plano de coordenadas con los nodos sensores que forman parte de la red que estamos simulando.

```
PC=new Plano2(10);
```

El valor que se pasa como argumento en la creción del plano se corresponde con el número de separaciones que queremos que aparezcan en el plano cartesiano donde se mostrará la topología de la red, de modo que bajo el plano cartesiano aparece una malla a modo

de regla. Estas separaciones toman el valor de 5 unidades de medida. Por tanto, el plano que se muestra al usuario tiene una dimensión de 50x50, con 10 separaciones verticales y 10 horizontales.

Una vez creado el panel, se añade a la ventana de la interfaz gráfica de la siguiente manera:

```
panelCanvas.add(PC, BorderLayout.CENTER );
panelActual.add(panelCanvas, BorderLayout.CENTER);
```

6.1.9 Método tablaDatos

Este método se invoca desde el método `panelDatosSimulacion`. En este método se crea una matriz a la que llamamos `datos`, y cuyos datos los recogemos de los argumentos que se introducen al principio de la simulación:

```
Object[][] datos = new Object[][] {
   {"Número de nodos", String.valueOf(avrora.Main.nodesNumber)},
      …
}
```

6.1.10 Método seleccionBotones

Como se ha explicado en métodos anteriores, se han desarrollado una serie de botones y se han activado los eventos de ratón para que el usuario pueda cambiar de ventana durante la simulación de la red de sensores, y así ver la información que necesita en cada momento. Para que esto ocurra, este método debe implementar la interfaz `ActionListener()` de Java.

En este método, dependiendo del botón que se haya seleccionado, se solicita que se muestre en la interfaz gráfica una pantalla u otra. Para esto, se almacena en una variable `JButton` el nombre de la pestaña que se quiere mostrar, y se actualiza el panel principal de acuerdo a la información solicitada:

```
JButton b = (JButton)e.getSource();

if((b.getText()).equals(b1.getText())){
      panelActual.setVisible(false);
      panelActual = panelDatosSimulacion();
      frame.add(panelActual, BorderLayout.CENTER);
      panelActual.setVisible(true);
}
```

Este código se repite para cada uno de los paneles diseñados, es decir, para “Datos Simulación”, “Tráfico y Energía”, “Despliegue Inicial” y “Despliegue Final”.

6.2 DataManagement.java

La clase `DataManagement.java` realiza todo el procesado de los paquetes que se transmiten por la red. Esta clase incluye los siguientes métodos que se describen seguidamente:

- `esFicheroCorrecto ()`

- `run()`
- `updateInfoNodes()`
- `modificarMatrizVecinos()`

6.2.1 Método esFicheroCorrecto

Cada uno de los paquetes que circula por la red se debe procesar de manera individual para extraer toda la información incluida. Por esto, el tráfico de la red se almacena en un fichero y es en este método en el que se accede al fichero origen en el que se guardan los paquetes y, uno a uno, se procesan. El método recibe como argumento el fichero a leer. Ya que es posible que el fichero esté corrupto, en primer lugar se realizan algunas verificaciones para comprobar que el fichero existe y es correcto:

```
if(!fichero.exists()){
      return false;
}
if(!fichero.isFile()){
      return false;
}
if(!fichero.canRead()){
      return false;
}
```

Si alguna de estas condiciones se cumple, se lanza un mensaje de error y se aborta la ejecución de la simulación. Por el contrario, si no se cumplen, se deja seguir con la simulación.

6.2.2 Método run

Este es el método principal de esta clase, y es el que se ejecuta cuando se invoca a `DataManagement` desde otras clases. En primer lugar, en esta clase se verifica la existencia y el acceso al fichero de información, invocando al método `esFicheroCorrecto` descrito anteriormente.

También es en este método en el que se establece una condición de parada del buffer de lectura. Esta condición se incluye para no encontrarnos con que no hay nada más que leer en el fichero y evitar la aparición de una excepción que aborte la simulación.

Por esto, el buffer de lectura estará activo, y se accederá al fichero en el que se vuelcan los resultados de la simulación de la red de sensores, siempre que el tiempo indicado en los parámetros de entrada como tiempo de simulación (`tEjecucion`) sea mayor que una variable, a la que denominaremos `hora`, y que se calcula sumando al tiempo relativo de la ejecución un parámetro `alpha` arbitratio, en este caso definido en 1.5 segundos. De este modo nos aseguramos de parar el buffer de lectura antes de que se termine la simulación, y por tanto, no existan paquetes que analizar.

```
do{
      hora = avrora.monitors.PacketMonitor.getHora() + alpha
      s=br.readLine()
      …
}
```

```
while(tEjecucion > hora);
```

Desde este método se llama a `updateInfoNodes` para analizar cada uno de los paquetes, el cual se describe a continuación.

6.2.3 Método updateInfoNodes

En este método se procesa cada línea representando cualquier tipo de paquete que circula por la red, se extrae su información, y se actualizan las matrices y contadores que se utilizan para la representación de la información en la pestaña de "Tráfico y Energía". También se analizan los paquetes para obtener los vecinos de cada nodo. En primer lugar se comprobará la longitud del paquete que se va a analizar. La longitud de los paquetes nos permitirá determinar de qué tipo es cada uno de los paquetes. Si la longitud de este paquete es menor de 89, nos encontramos ante un paquete ACK. Identificamos el paquete y actualizamos el contador de paquetes ACK que han circulado por la red:

```
numPaquetesACK=numPaquetesACK+1;
```

Si la longitud del paquete es mayor, nos encontramos ante un paquete de beacon o de datos. Para seguir con el tratamiento, se extrae de cada uno de los paquetes su nodo origen y su nodo destino, de la siguiente manera:

```
int nodoDestino =
Integer.parseInt((cadena.substring(78,80)+cadena.substring(75,77)
),16);

int nodoOrigen =
Integer.parseInt((cadena.substring(84,86)+cadena.substring(81,83)
),16);
```

Los valores están en hexadecimal, por lo que se realiza una conversión a entero. Si al realizar la conversión de cualquiera de los nodos, el valor que obtenemos es de 65535, nos encontramos ante un paquete tipo *broadcast* (es decir, un paquete que se envía a todos los nodos de la red).

También extraemos si el paquete que se está analizando se está enviando o recibiendo. Para ello, miramos si aparece el valor "<" en la cadena. Este valor indica que el paquete se está recibiendo. En caso contrario, si aparece el varlo ">" quiere decir que el paquete se está enviando.

```
enviando = cadena.substring(29,30).equals(">");
```

A continuación, comprobamos si el paquete correspondiente a línea que se está analizando es un paquete de beacon (paquete con información de la topología de red). Tal y como se ha explicado en apartados anteriores, los paquetes beacon son de tamaño variable y se identifican por contener el valor "18" en las posiciones 87 y 88, por lo que usamos estas dos posiciones de la cadena, de la siguiente manera:

```
if (cadena.substring(87,89).equals("18")){
…
}
```

Si esta condición se da, es debido a que se trata de un paquete *beacon*. De estos paquetes se obtiene la información de las vecindades de los nodos, por lo que se pasa también la cadena al método `modificarMatrizVecinos` (método que se desarrollará más adelante) para la actualización de esta información en la matriz correspondiente:

```
MatrizVecinos = modificarMatrizVecinos(cadena,
                            nodoDestino, nodoOrigen, enviando);
```

Por el contrario, los paquetes de datos se identifican porque las posiciones 87 y 88 toman el valor "17", por tanto:

```
if (cadena.substring(87,89).equals("17")){
…
}
```

Dependiendo de si el paquete se envía o se recibe, y con la información obtenida al principio del método a cerca del nodo origen y nodo destino, se actualizan las matrices correspondientes de envío y recepción de paquetes de datos.

Hay paquetes que no cumplen ninguna de las condiciones que se han explicado anteriormente. Todos estos paquetes son confirmaciones a los paquetes de datos, es decir, paquetes ack.

6.2.4 Método modificarMatrizVecinos

A este método se le pasa como argumentos de entrada el contenido del paquete (payload), el nodo origen y el nodo destino, y si el paquete es enviado o recibido.
En primer lugar, se calcula el número de vecinos que tiene el nodo. Cada vecino se representa con 3 bytes más que se añaden al final del payload: los dos primeros son el identificador del nodo vecino y el último byte representa la calidad de servicio del enlace entre el nodo y el vecino:

```
int numVecinos = Integer.parseInt(payload.substring(90,92), 16);
```

Si el nodo receptor toma el valor 65535 nos encontramos ante un paquete broadcast. Tendremos en cuenta los paquetes que no se han emitido en modo broadcast, es decir, una comunicación punto a punto. Aplicando la siguiente fórmula, conseguimos obtener qué nodos son los vecinos del nodo receptor, independientemente del número de vecinos que se indiquen en el paquete beacon. Una vez obtenido el valor del nodo, se actualiza la matriz. El siguiente código muestra esta funcionalidad:

```
for (int i=0;i<numVecinos;i++) {
String VecinoCadena = payload.substring((111+9*i),(113+9*i))+
                payload.substring((114+9*i),(116+9*i));
int vecino = Integer.parseInt(VecinoCadena,16);
MatrizVecinos[sendingNode][vecino] = 1;
```

Tal y como se ha explicado al comienzo de este método, cada vecino se representan con 3 bytes que se añaden al final del paquete (2 bytes para identficar al nodo vecino más un tercer byte para establecer la calidad de servicio del enlace (QoS, ***Q**uality of **S**ervice)*. El formato de los paquetes de beacon establece que el primer nodo vecino aparece a partir de la posición 112,y los nuevos vecinos aparecen cada 9 bytes después de esta posición. El nodo vecino se extrae en

modo hexadecimal, por lo que es necesario convertirlo a un valor entero antes de poder modificar la matriz de vecinos.

Si hay comunicación entre los nodos, el valor de la matriz que se corresponda con el nodo del que se recibe el paquete beacon (fila) y el vecino (columna) se establece a `1`. El resto de celdas de la matriz en las que no haya habido comunicación, tendrán un valor de `0`.

6.3 PacketType.java

`PacketType` se usa únicamente para el establecimiento e inicialización de las variables usadas en la clase `DataManagement`. Esta clase recibe como argumentos el número de paquetes de ACK, beacons y paquetes de datos.

```
public PacketType(int numACK, int numData, int numBeacon) {
      this.numACK = numACK;
      this.numData = numData;
      this.numBeacon = numBeacon;
      }
```

6.4 Plano2.java

Esta clase es la utilizada para la representación de la red de sensores que se puede ver en la pestaña "Despliegue Inicial". En esta clase se implementan los siguientes métodos:

- `Plano2 ()`
- `paint()`
- `dibujarPuntos()`

6.4.1 Método Plano2

Este método es el constructor de la clase. Por parámetro se ha de pasar un entero que definirá el número de marcas o separaciones que tendrá el plano. Una vez establecidas las separaciones, se pasa a leer el fichero de topología que se indica mediante un parámetros al inicio de la simulación, y que incluye la localización de los nodos en el espacio. Para ello, es necesaria la creación de una instancia `FileReader` y `BufferedReader` para leer de este fichero, que se crea de la siguiente manera:

```
FileReader fr=null;
try {
       fr = new FileReader(avrora.Main.ficheroTopologia);
      BufferedReader br = new BufferedReader (fr);
      String linea;
 }
```

6.4.2 Método paint

Para representar los nodos especificados en el fichero, se deberá ejecutar el siguiente fragmento de código. Hay que tener en cuenta que si el nodo que se va a representar es el nodo 0

o gateway, se representará en color rojo, mientras que el resto de nodos sensores que intervienen en la simulación se representan en color azul.

```
for(j=0; j<nodos; j++) {
     if (j==0){
      g.setColor(Color.RED);
      Y=(int) ((-miny)/(maxy - miny)*(MaxY))-3;
      X=(int) ((-minx)/(maxx - minx)*(MaxX))-3;
     }else{
      g.setColor(Color.BLUE);
      X=this.getWidth()/2+(cx[j]*MaxX/100)-3;
      Y=this.getHeight()/2-(cy[j]*MaxY/100)-3;
     }

     g.fillOval(X, Y,  12, 12);
     g.drawString("A"+(j)+"("+cx[j]+","+cy[j]+")", X,Y);
}
```

6.4.3 Método dibujarPuntos()

Desde este método se invoca al método `paint`, definido en el apartado anterior. Se toma del fichero de topología que se pasa por parámetro en la ejecución de Avrora la ubicación de los nodos y se representan en el plano.

```
this.paint(this.getGraphics());
```

7. Conclusiones

En este capítulo se revisan y evalúan los objetivos del proyecto que se establecieron al principio del mismo y se describieron en el capítulo 1. También se incluyen las posibles líneas de desarrollo de este proyecto, así como un análisis del presupuesto.

7.1 Revisión de los objetivos

Al comienzo de este proyecto, en el capítulo 1, se definieron una serie de objetivos que se pretendían cumplir una vez finalizado el mismo. A continuación se pasa a describir el grado de consecución de cada uno de ellos:

- **Control de los parámetros de entrada de la simulación**.

Esta funcionalidad se ha incluido en la interfaz gráfica desarrollada en este proyecto. Se ha dedicado una pestaña dentro de la interfaz gráfica, denominada "datos de la simulación", en la que se pueden ver cuáles son los parámetros más relevantes que se han indicado para poder simular la red de sensores en Avrora, tales como número de nodos usados, paquetes que se quieren monitorizar, la topología de red que se ha usado, etc.

- **Simulación de las aplicaciones y análisis del tráfico.**

La simulación es algo que nos proporciona Avrora, pero no el análisis de tráfico. Este objetivo se ha conseguido mediante la inclusión de una clase programada en java a la que se ha denominado *DataManagement.java*, con la que se ha logrado que se extraiga toda la información relevante de los paquetes que circulaban por la red. Los datos obtenidos sobre la cantidad de tráfico que ha cursado cada nodo, así como las vecindades que se establecían entre los nodos según avanzaba la simulación, se han almacenado en variables globales que pueden ser usadas desde otras clases.

- **Monitorización del consumo energético del módulo de radio de los nodos.**

La energía se calcula en uno de los monitores que están incluidos por defecto en el desarrollo de Avrora. Este objetivo se ha completado incluyendo en una de las pestañas de la interfaz gráfica el valor de energía total consumida durante la simulación tanto en transmisión como en recepción para cada uno de los nodos.

- **Desarrollo de un módulo en Java que permita la representación gráfica de los valores relevantes de la simulación.**

El diseño de este módulo ha sido el principal objetivo de este proyecto. Tras la implementación del módulo AvInterfaz, asociado a Avrora, se ha conseguido representar en una ventana con varias pestañas el análisis de los paquetes de datos, beacon y de confirmación (ACK) que circulan por la red, la energía consumida por el módulo de radio, la topología de la red tanto al principio de la simulación cuando no hay conexiones entre nodos, como al finalizar la misma, indicándose el camino óptimo que siguen los paquetes desde los nodos hasta llegar al gateway.

7.2 Trabajos futuros

En este apartado se incluye una serie de líneas de mejoras, no incluidas en este proyecto, y que pueden servir como base para la elaboración de futuros proyectos fin de carrera relacionados con esta temática.

El simulador Avrora puede simular varios monitores a la vez, sería interesante que se ampliara la interfaz gráfica con el estudio de algunos de ellos

- **Análisis de nuevos monitores: *Memory Monitor*.**

  El monitor de memoria (Memory Monitor) recoge información sobre cómo es el acceso a la memoria de los nodos sensores durante la simulación. Acumula el número de veces que se ha accedido a la memoria y el estado de ocupación de la misma, resultado que se puede ver al final de la simulación. Se podría incluir esta información en la pestaña de Tráfico y Energía diseñada en este proyecto, o bien crear una nueva pestaña dentro de la interfaz gráfica dónde, por cada nodo, se pudiera comprobar el número de accesos y la memoria interna del nodo sensor que está disponible.

- **Desarrollo de nuevos monitores en Avrora.**

  Avrora es un simulador de código abierto, por lo que es posible desarrollar nuevas funcionalidades sobre el código ya existente. De hecho, existen monitores aún no programados en Avrora (aunque sí enunciados) y cuyo análisis sería muy interesante, tal y como el SimpleBatteryMonitor. Este monitor, a desarrollar, apagaría un determinado nodo de la red de sensores cuando supere un umbral de energía previamente definido. La inclusión de este monitor implicaría modificar la representación gráfica de la topología de la red resultante de la simulación, para que mostrara los nodos apagados por el sistema para evitar problemas de transmisión en la red.

- **Mejoras en la interfaz gráfica actual.**

  Dentro de la interfaz gráfica, se podría introducir una nueva pestaña en la que se mostrara cómo es el tráfico que se está cursando por la red en tiempo real. Esto permitiría observar, antes de que finalice la ejecución, qué nodos son los que soportan más tráfico y por tanto, los más problemáticos en caso de pérdida de conectividad con ellos o agotamiento temprano de las baterías.

7.3 Presupuesto

Una parte fundamental en la gestión de proyectos es el estudio del presupuesto. Los presupuestos nos indican la partida económica que se ha de destinar para poder completar el proyecto. En este caso, se van a detallar los gastos necesarios del proyecto, divididos en tres grupos:

- **Recursos humanos**: personas que son necesarias para el desarrollo del proyecto.
- **Recursos materiales**: soporte informático o no, necesario para el desarrollo del proyecto.
- **Servicios subcontratados**: partida destinada a la elaboración de tareas del proyecto por parte de empresas externas.

A continuación, se detallarán cada uno de estos tres apartados:

7.3.1 Recursos humanos:

Para calcular el coste total que implican los recursos humanos para el desarrollo del proyecto, se tendrá en cuenta cada una de las fases que lo forman, es decir:

- Análisis del entorno del proyecto y aprendizaje del estado del arte.
- Análisis del problema.
- Diseño.
- Implementación.
- Evaluación.
- Documentación.

Además, se tendrán en cuenta las siguientes consideraciones:

- El proyecto ha sido desarrollado por una única persona (recurso).
- Se han dedicado una media de 20 horas semanales.
- Se ha trabajado un total de 38 semanas, o lo que es lo mismo, 760 horas.

Se estima que la retribución media por hora de trabajo está establecida en 35€. En la siguiente tabla se muestra para cada fase del proyecto, la distribución de las horas empleadas para la finalización del mismo:

	Horas dedicadas	Coste/hora (€)	Coste total (€)
Análisis del entorno	80	35	2.800
Análisis del problema	80	35	2.800
Diseño	180	35	6.300
Implementación	140	35	4.900
Evaluación	120	35	4.200
Documentación	160	35	5.600
Total:	760	35	**26.600**

Tabla 21 - Costes asociados a los recursos humanos

7.3.2 Recursos materiales

En este punto, se tendrán en cuenta los recursos hardware necesitados para el desarrollo de este proyecto. En este caso, se contabilizará el ordenador portátil utilizado para la realización del mismo.

A continuación se adjunta la tabla con los costes asociados al uso de los recursos materiales:

	Coste (€)	% uso dedicado	Tiempo dedicado	Periodo de depreciación	Coste imputable
Ordenador portátil	750	100%	9,5 meses	60 meses	**118,75 €**

Tabla 22 - Costes asociados a los recursos materiales

Sólo se ha de tener en cuenta el coste imputable que implica el uso del portatil. Teniendo en cuenta que el equipo se dedica durante el desarrollo del proyecto el 100% del tiempo al desarrollo del mismo, y considerando un periodo de depreciación del ordenador de 60 meses, se obtiene un coste imputable de 118.75€.

El coste imputable se obtiene mediante la siguiente fórmula:

$$Coste\ imputable = \frac{A}{B} \times C \times D$$

donde:

- *A= nº de meses desde la fecha de facturación en que el equipo es utilizado*
- *B= periodo de depreciación*
- *C= Coste del equipo (sin IVA)*
- *D= % del uso que se dedica al proyecto*

7.3.3 Servicios subcontratados

En este apartado contabilizan los costes asociados a la contratación de la conexión a internet usada para la búsqueda de información y la comunicación con el tutor del proyecto. De igual manera, en este punto se tienen en cuenta los costes asociados a la impresión del documento. En la tabla adjunta se detalla toda la información:

	Cantidad	Coste (€)	Coste total
Conexión a internet (coste mensual)	9,5	35	332,5
Impresión de documentos	2	60	120
Total			**453**

Tabla 23 - Costes asociados a los servicios subcontratados

7.3.4 Costes totales

Los costes totales se calculan sumando todos los costes anteriores. En la siguiente tabla se muestra la suma de todos ellos:

	Coste
Recursos humanos	26.600
Recursos materiales	119
Servicios subcontratados	453
Total	**27.172**

Tabla 24 - Costes totales

7.3.5 Evaluación personal

He estado vinculada al mundo de las telecomunicaciones desde el inicio de mi vida universitaria, compaginando mis estudios de Ingeniería Técnica en Telecomunicaciones con el mundo laboral. En la relación que he tenido en mi trabajo con informáticos he comprendido la importancia de una buena interfaz gráfica con la que representar la información.

Este hecho hizo que me decantara por un proyecto final de carrera del área de informática, con el que implementar una interfaz de representación de datos. He conseguido rescatar la programación en Java y aplicarla en la simulación de redes de sensores inalámbricos, consiguiendo unificar las telecomunicaciones, que son mi pasión, con la informática.

Bibliografía

01. (s.f.). *https://www.fas.org/irp/program/collect/sosus.htm*.

02. (s.f.). *http://www.cs.berkeley.edu/~culler/AIIT/papers/historical/Lacoss%20DSN%201986.pdf*.

03. (s.f.). *http://robotics.eecs.berkeley.edu/~pister/SmartDust/*.

04. (s.f.). *http://www.sunspotworld.com/*.

05. (s.f.). *http://www.art-of-technology.ch/en/reference-projects/bt-node*. Obtenido de imagen: http://www.btnode.ethz.ch/static_docs/doxygen/btnut/btnode_rev3.20_golden_crop.jpg.

06. (s.f.). *www.twist.tu-berlin.de/wiki/WSN/EyesIFXv1Node*. Obtenido de imagen: http://docs.tinyos.net/tinywiki/images/thumb/5/59/EyesIFX.jpg/140px-EyesIFX.jpg.

07. (s.f.). *http://wsnblog.com/tag/imote2-multimedia-board/*. Obtenido de imagen http://ait.upct.es/saeta/images/5/5a/Imote2_4.jpg.

08. (s.f.). *http://wsnblog.com/tag/iris/*. Obtenido de imagen http://www.zdnet.com/i/story/60/01/035409/sensor060710a.jpg.

09. (s.f.). *http://blog.memsic.com/mica2_mote/*. Obtenido de imagen http://logicalneighbor.sourceforge.net/images/mica2.jpg.

10. (s.f.). *https://www.eol.ucar.edu/rtf/facilities/isa/internal/CrossBow/DataSheets/mica2dot.pdf*. Obtenido de imagen http://www.comsys.rwth-aachen.de/uploads/pics/mica2dot.jpg.

11. (s.f.). *http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf*. Obtenido de imagen http://www.cmt-gmbh.de/Produkte/WirelessSensorNetworks/Images/MICAz_gross.jpg.

12. (s.f.). *http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=252*. Obtenido de imagen http://openwsn.berkeley.edu/browser/trunk/docs/multimedia/telosb.jpg?format=raw.

13. (s.f.). *http://www.tinynode.com/*. Obtenido de imagen http://www.snm.ethz.ch/snmwiki/pub/uploads/Projects/tinynode.jpg.

14. (s.f.). *http://wsnblog.com/tag/eko/*. Obtenido de imagen http://bullseye.xbow.com:81/Products/Product_images/Wireless_images/eko_basic_kit_sm.jpg.

15. (s.f.). *http://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CDMQFjAA&url=http%3A%2F%2Fwww.teo.unt.edu%2Fcri%2Fppt%2Fwsn090828.pptx&ei=iPt0UcKk*

F8PEPeDlgIgM&usg=AFQjCNHQZq5xdA-Z2WeXYL-V2We59zIB4g&bvm=bv.45512109,d.d2k&cad=rja. Obtenido de

16. (s.f.). *http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/MDA320_Datasheet.pdf*. Obtenido de imagen: http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/MDA320_Datasheet.pdf.

17. (s.f.). *http://www.xbow.jp/mts_mda_datasheet.pdf*.

18. (s.f.). *http://www.xbow.jp/mts_mda_datasheet.pdf*.

19. (s.f.). *http://www.xbow.jp/mts_mda_datasheet.pdf*.

20. (s.f.). *http://www.investigacion.frc.utn.edu.ar/sensores/Equipamiento/Wireless/MTS-MDA_Series_Users_Manual.pdf*.

21. (s.f.). Obtenido de http://www.thermofisher.com.au/Uploads/file/Environmental-Industrial/Process-Monitoring-Industrial-Instruments/Data-Acquisition/Wireless-Sensors/Memsic/MIB510-SERIAL-INTERFACE-BOARD.pdf

22. (s.f.). *http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=227*.

23. (s.f.). *http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=179*.

24. (s.f.). *http://www.devisersoftware.com/down/20114650.pdf*.

25. (s.f.). *http://es.wikipedia.org/wiki/IEEE_802.15.4*.

26. (s.f.). *http://www.zigbee.org/*.

27. (s.f.). *http://www.tinyos.net/*.

28. (s.f.). *http://nescc.sourceforge.net/*.

29. (s.f.). *http://www.contiki-os.org/*.

30. (s.f.). *http://dunkels.com/adam/pt/*.

31. (s.f.). *http://rosejn.net/publications/mantis-wsna.pdf*.

32. (s.f.). *http://dl.acm.org/citation.cfm?id=1372737*.

33. (s.f.). *www.java.com/es/*.

34. (s.f.). *http://compilers.cs.ucla.edu/avrora/*.

35. (s.f.). *http://www.advanticsys.com/services/energy-efficiency-monitoring-solution/?lang=es*.

36. (s.f.). *http://www.advanticsys.com/services/agricultural-environment-monitoring/?lang=es*.

37. (s.f.). *http://home.etf.rs/~vm/ppt/1%20A_Survey_Military_Apps_WSNs_MPDj.ZT.GD.VM.pdf*.

38. (s.f.). *http://www.trackss.net*.

39. (s.f.). *http://www.radioptica.com/sensores/*.

40. (s.f.). *http://www.minetur.gob.es/industria/observatorios/SectorElectronica/Actividades/2009/Federaci%C3%B3n%20de%20Entidades%20de%20Innovaci%C3%B3n%20y%20Tecnolog%C3%ADa/FEDIT%20RedesSensoresEdificios.pdf*.

41. (s.f.). *http://codeblue.com/*.

42. (s.f.). Obtenido de http://administracionelectronica.gob.es